MUG編／コンピュータ言語

MUMPS入門

若井一朗訳

Mumps Primer

R.F.Walters
J.Bowie
J.C.Wilcox

日刊工業新聞社

MUG 編／コンピュータ言語

MUMPS 入門

R. F. Walters
J. Bowie　　　　著
J. C. Wilcox

若井一朗 訳

日刊工業新聞社

Mumps Primer　REVISED, 1979

R. F. Walters

J. Bowie

J. C. Wilcox

この MUMPS 入門書 (MUMPS Primer) は，1975年6月13日付 MDC 文書番号 1/11 による M. E. Johnson and R. E. Dayhoff 著 MUMPS Primer を，1977年9月の ANSI XII. 1-1977 の標準 MUMPS 言語仕様制定化にともない，MDC/MUG の文書改訂委員会が標準 MUMPS に合致する改訂をおこなったものである.

日本語版への序文

MUMPS Primer 改訂版の日本語訳が出版される運びとなり喜びにたえない．日本にどれだけ多くの MUMPS 言語利用者がいるか，そしていかに高度な活動が展開されているかを私たちは熟知しているだけに，今後さらに MUMPS 言語が普及されることにわずかでも役立ちうると考えるだけで光栄至極である．

この MUMPS Primer の改訂版は Jerome Wilcox と Jack Bowie および私が MDC の Primer 初版を注意深く検討して書き改めたもので，私たちの主な目標はこの入門書をしてプログラミングの初心者にも，経験を積んだシステムアナリストにも MUMPS 言語の特性をよく把握させるという点にあった．このため，この入門書の概要を説明する導入の章をもうけて初心者ないし経験者による効果的な本書の利用法を述べておいた．基本的な土台作りに始まって高度な概念に至る迄を，論理的に章を追って展開した試みが本書を読み易さと明解さの点で，改良しているものと思う．

本書の最後に盛り込もうとして選んだ2つのプログラム例は MUMPS プログラムの諸相を一覧できるように書かれたもので，このまま動かすには，実際にはエラーの生じることもありうる．これらの構文上または論理上の未完部分は読者が実際に MUMPS 機種で見出し手を加えて完成させることが望ましい．

元来，本書は MUMPS 機種の提供者ないし言語装備者が，彼らの装備に特有な，標準 MUMPS に付け加えた諸機能について読者のために一章を追加して彼らの言語のユーザーに勉強させることを意図しているが，日本でもそうであるように言語装備者が多数出現しつつある現在，メーカーを限って一章の追加を依頼することの困難性から，出版の段階ではこれをなしえなかった．しか

し，読者はその用いる機種に特有の拡張機能や，装備依存的な部分を簡明に要約するか，出来ればメーカーにこれらの要約を作ってもらって本書の終りに貼りつけて，本書を読者自身のものに仕立てることを強く勧めたい．

また，私たちの心からの願いは読者が本書を通じて本書に欠けている面をMUMPS 言語をも含めて活発に討論し，これらで改良できる点をそれぞれ示していただきたいことである．これ迄日本マンプス ユーザーズ グループや，そこから北米のマンプス ユーザーズ グループに参加した多くの活動家たちとの私の交流で得た経験は，私たちの努力が注意深く日本で分析されたあげく，どれほど私たちに利益をもたらしてくれるものであるかを如実に立証している．

この入門書の読者たちが，いつの日か米国を旅する際にカリフォルニア大学の Davisキャンパスを訪問先に含めて下さり，めいめいの仕事やMUMPS観をお聞きする機会と同時に本書の改良点などについての教えを与えて下されば何よりの幸せである．また，本書の日本語完訳によって私に与えられた光栄に深く感銘し，日本の MUMPS ユーザーと今後永らくより緊密な関係をもちうることを心から期待したい．

1980年2月20日

カリフォルニア大学教授
Richard F. Walters, Ph.D.
Departments of Community Health and Electrical Engineering

訳 者 序 文

標準 MUMPS の仕様の第一次制定がなされてから2年余の今日，既に世界で 20 種以上の機種に標準 MUMPS が装備されている．ミニコンから出発したこの言語は汎用大型コンピュータやマイクロ・コンピュータにも装備されはじめた．我が国では昭和 51 年に通産・厚生両省の共管になる（財）医療情報システム開発センターが中心となってカナ文字を含む標準 MUMPS の装備を開始して以来既に国産機種のみで7社が MUMPS の装備を発表している．

日本マンプス ユーザーズ グループが昭和 50 年に発足した時に，平川顕名会長が出席者 200 名に提供した資料が Martin E. Johnson と Ruth E. Dayhoff による MDC 資料 MUMPS Primer であったことを想起したい．これは昭和 51 年3月（財）医療情報システム開発センターが日本語訳を限定部数の資料として公開し，その日本語版は昭和51 年いらい当時の日本 MUG 会長野村裕博士のお骨折りにより，日本マンプス ユーザーズ グループの講習会テキストとして4年間利用させていただいたものであった．このテキストは，現在では多くの個所で標準 MUMPS 仕様に合致しないところがあり，またバランストトリーによるグローバル ファイルの出現によって，物理的構造上ファイルの設計法にそれほどユーザーが苦慮する必要がなくなったことなどから，すでに2～3年前から米国 MDC/MUG では改訂の要が叫ばれていた．

本書は名著 MUMPS Primer の，R. F. Walters らによる 1979 年度改訂版である．これを読むにつれ，その初版にみられた練習問題を多く含んだ系統的な教科書の味わいがすっかり無いことに奇異な感をもつのであるが，その代り，標準 MUMPS の入門書として，その概念に読者を徐々に深入りさせていく手法は，自らを MUMPS 端末に向かわせ，プログラミングの技法の習得をさせるワーク ショップではないにしても，プログラミング言語としての MU-

MPS に著しい興味を抱かせずにはおかない入門書としては，ユニークな力作であることが解る．日本語による標準 MUMPS への入門講座は日本マンプス ユーザーズ グループの昭和53年度（第5回大会）および54年度（第6回大会）において，それぞれ新規に日本マンプス ユーザーズ グループ会員講師による講習会テキストが出ており，日本マンプス ユーザーズ グループから入手ができる．また，市販されている標準 MUMPS 教科書としては，若井らの翻訳になる「MDC 編，標準マンプス言語マニュアル (MUMPS Programmers' Reference Manual)，コロナ社版」がある．

MUMPSは既に昭和49年 DEC 社の MUMPS-11（標準化以前の MUMPS 方言）にカナ文字が装備され，カナ文字のパターン コードと共に日本語の文字列処理も便利になっていたが，現在では更に次期準備仕様としての確実性のある文字列添字をカナ文字で装備しはじめる姿勢が3～4社に見られており，カナ日本語キーや，さらにグローバル内に数千字の漢字・ひらがなのフォントイメージを入れた日本語の全文字処理等も標準 MUMPS に付加される時代となっている．本書では少なくとも日本で用いる MUMPS 機種に装備されているカナ文字をプログラム例の中でできるだけ多く用いることにした．また日本 MUG の派遣する MDC 委員がつたえる改訂されることの決った ANSI 標準規則にもとづき1年古い本書の説明を邦訳で改編したところもある．

今後さらにこの，FORTRAN，COBOL，PL/I と並んで，標準化され，しかも移送性，互換性を強く標準化の根底にふまえた MUMPS の利用度が高まり，尚も数多くの入門書や上級技術テキストが出現することを期待すると同時に，MUMPS 利用の普及に多大の努力をかたむけられた歴代の日本 MUG 会長に深く敬意を表する．また，本書の短期間での翻訳にあたって助手をつとめて下さった京都大学工学部情報工学科の三上斉君に深く謝意を表する．

昭和55年5月20日

若 井 一 朗

目　　次

第 1 章

はじめに

MUMPS とは何か?

コンピュータに関する数多くの概念を説明する際に MUMPS という用語がよく用いられている. MUMPS とは Massachusetts General Hospital's Utility Multi-Programming System の頭文字をとったものであるが, 1967 年にその概念ができてからこの用語は種々の意味を含むようになった.

MUMPS には 3 つの概念が包含されているとみてよい. 第1に解釈語としてのコンピュータ言語である. "解釈語 (インタープリタ)" というのはコンパイル言語でおこなう "オブジェクト コード" に変換する作業を必要とせず直接実行できる点にある. 第2に MUMPS は医療用の データ ベース のような大きいファイルの数々を ダイナミック に扱う能力のあるところから, "データ ベース マネジメント システム (DBMS)" のために設計された多くの言語に匹敵するファイル マネジメント システムとしての能力をもっていると考えてよい. 第3に MUMPS は伝統的に小型コンピュータに単一言語システムとして装備されてきたので, MUMPS という用語にはオペレーティング システム (OS) の概念も含まれてきた. 解釈語とファイル マネジメントの面は不可欠の要素であるが OS としての要素は必ずしも MUMPS に固有の性質ではない.

次の節で MUMPS 言語の歴史と, MUMPS に関する種々の見方を紹介するが, その背景はともかくとして, MUMPS は今や米国国家規格協会(ANSI)に認可された 4 つの言語の 1 つとなり, マイクロ プロセッサから ミニコンピュータそして大型コンピュータにまで装備されるようになった. そしてすでにヨーロッパ 30 ヵ国や日本にもよく普及し, その利用者の中には驚異的な数の企業団体がある.

MUMPS 小史

タイム シェアリング システムのうえでコンピュータと対話をおこなう構想は1960年代のはじめ，すでにはじまっていた．1967 年の後半 Massachusetts General Hospital の Laboratory of Computer Science は医療の領域では対話型システムが有効であることを認め，ミニコンピュータで動く対話型タイムシェアリング環境でのファイル マネジメント システムをもつ文章型の文字列を操作する能力を備えたコンピュータ言語の開発をはじめた．この言語についての最初の記述は 1969 年秋期ジョイント コンピュータ カンファレンスで発表された[1]．設計者らが目標としたことが達成できたというこの成功によって，この言語はすぐに臨床医学の場と医学教育の場で普及しはじめ有名となった．その名声は創始者の機関だけでなく，他の多くの機関にも拡がり，またたく間に多くの MUMPS 方言が提供されるまでになった．これらは MUMPS 原語を機能的に拡張したり，変更を加えたものでそれぞれ改良を意図したものではあったが，これらや，あるいは MUMPS 原語を用いて新しい応用プログラムが開発されるにつれて大きい混乱が生まれるようになった．そして 1972 年にはついに 14 種を下らない独立した稼動中ないし開発中の MUMPS 言語の版がみられるようになった．

MUMPS の大きい可能性が認識されたため 1972 年には，米国標準局(NBS)と米国保健・教育・福祉省の分庁である国立保健サービス研究センター (NC-HSR) をスポンサーとして新しい標準 MUMPS 言語の開発に向けての共同作業がはじまった．この作業には既存の方言 MUMPS とその装備者やユーザーも参加した．この作業はおしつけでなく同意と歩みよりを基調とするプロジェクトであった．言語の定義を厳密に定めるためのこの MUMPS 開発委員会 (MDC) は構成の形体と規約をととのえていかなる見解をも公平に考慮できるようにした．会合は 1973 年から 1975 年まで年に数回おこなわれ，それまでに新しい標準 MUMPS 言語が同意のうえで採択された．この言語定義は 1976 年の春 ANSI に認可申請されて，ANSI の規定したキャンヴァス法を経過し

たのち，異議なく1977年9月に ANSI 認可となった．

この時点から標準 MUMPS は種々のシステムに装備されるようになった．今ではマイクロ コンピュータにも，ミニコンピュータにも，さらには大型の中央電算機にも装備されるようになり，その利用は医療，教育，企業の中に拡まっている．言語に対する興味も幅を拡げ，その装備の数も月と年を経るに従って増しつつある．

MUMPS 開発委委会は言語の定義の反復点検と今後の拡張に関してひきつづきその責任をもっており，少なくとも年間2回の会合をもって将来の拡張と現在の定義の改版を進めている．MUMPS 開発委員会で正式に採択された改変部分は ANSI 規定で義務づけられている1982年改訂版の内容として提案されることになる．それまでの間 MUMPS 開発委員会とその言語の応用面を追究する相棒の MUMPS ユーザーズ グループは，この言語の応用によるソフトウエアの仕様書の慣習を完成させることに努力を傾けている．MUMPS に関するさらに詳しい資料の必要な向きは MUMPS ユーザーズ グループ代表者に申込むことをすすめる．

本入門書の利用法

この入門書（MDC/MUG による第2版）は MUMPS 言語の初入門者を対象としている．プログラミングについて多少の経験があれば学習の助けにはなるが，この入門書はプログラミング言語の入門講座での副読本として MUMPS 概念を教えるのにも役立つように構成されている．このゆえに直ちに他言語と異なった MUMPS の特徴的な部分の解説をするのでなく，単純な例を用いて基本的なプログラミングの方法を紹介しようとしている．

以上のような方針をとりながらも，この入門書はすでに MUMPS で経験のある読者にも利用価値がある．次の節では上級プログラマが拾い読みをして役に立つ利用法が述べてある．

初級入門者はこの入門書を標準 MUMPS 装備のコンピュータを用いながら学習することを望むだろう．理想をいえば，初心者は本書の順に本文と例題の

まとまりごとに進みながらときどき実際の言語を動かす時間をもつのがよい．同輩や教師の指導も，小量ずつながら役に立つものであるが，自ら，積極的に学習に参加して，助力を求める前に困難な課題と取りくむ学習体験が向上のもととなる．すでに学習した部分を復習することもほとんどの学習者に必要なことであり，現在のテーマ以外の部分を記憶しておくのに役立つはずである．

ある点までくると学習者は自らが興味をもつ分野の概念によって応用プログラムを考えてみるのが有益である．このプログラムを参照しながら，MUMPSの構造をより確かに理解していくことが，要約としてはやさしく理解できても特定の応用面で問題が提起される場合が多いので，すすめられる方法である．

初級および中級の学習者が本書を利用する際にも，標準 MUMPS を実際に動かすときにも，座右として用いる参考書が必要となる． これは「標準 MUMPS ポケットガイド」で MUMPS の主要構成要素を圧縮して便利な参照用手帳としたものである．これは実際のプログラミング時の時間を節約するのに役立つので，参照用に本書の中に引用挿入してある．

この入門書をさらに有用なものとするために，特に2つの章を設けた．これらは実際の応用面の感覚をユーザーがもつ助けとして，経験者の設計し装備した典型的な応用プログラム例である．これらは学習が進む段階で何度も研究してみるとよい．最終表として余白を設けたのは読者が利用するシステムの提供者が標準仕様以外に拡張した機能や，直接モードの命令など，装備特有の機能をメモして，実際のプログラミングに十分役立てるためである．

プログラミング経験者のための MUMPS 学習

他の言語でプログラミングの経験がある学習者は，この入門書を余りに単純すぎるとみるかもしれないが，標準 MUMPS に含まれた概念を知らない者にとって有用な部分があるのでそれについて説明しておく．

第2章では定文字列 (string literal) やダイナミック変数の記述があり，これらは FORTRAN や COBOL ではみられないもので， BASIC のある方言では同種のものをとり入れているものがある．第3章で説明する算術優先順位

は FORTRAN と異なっているので，コンピュータによる算術結果がエラーを出さないために必ず理解しておかねばならない部分である．

MUMPS の主要な機能の 1 つに文章型の文字列を操作する能力がある． これらの操作法については第 5 章と第 13 章に記述されている．MUMPS の配列はダイナミックにつくられ，また分散配列の形をとるので，特に注意が必要である．第 10 章に添字付きローカル配列について，第 12 章にグローバル変数の概念が記述されている．

プログラム コードを自律的に変化させる言語構造である 間接指定は 他のほとんどの言語のプログラマには未知なものであり，この概念は第14章に記述されている．

第 16 章に MUMPS 言語の将来についての展望を解説したあと，第 17，18 章に多くの典型的な MUMPS プログラミング技術を例示する 2 つの応用プログラムを用意した．

MUMPS 学習資料の請求先

MUMPS 言語に関する鍵となる参考書が MUMPS ユーザーズ グループから数種出されている．「ANSI 標準 MUMPS (X11.1)」と「標準マンプス言語マニュアル」(MDC 刊，MUMPS Prgrammers' Reference Manual，若井一朗ほか訳，コロナ社発行，1977) は MUMPS の真摯なユーザーにとってもっとも重要な 2 冊である．これらや他の資料を得るために請求する宛先は，

〒 461 名古屋市東区布池町 32 MUG ビル

日本マンプス ユーザーズ グループ事務局 電 052-936-5660

である[訳注]．

MUMPS 言語と他のプログラミングシステムとの比較に興味のある人々には，T. Munnecke らの論文[2] がよい参考となるであろう．またさらに上級の MUMPS プログラミング技術へと進みたい者には Beckley と Bridger の著

訳注） 日本 MUG 刊の標準 MUMPS 講習会資料もここから入手できる．

書[3] が有力な助けとなるであろう.

初心者に MUMPS 言語をコンピュータ ビデオ端末から対話型で教授するプログラム パッケージが J. Zimmerman ら[4]によって標準 MUMPS 言語で書かれている. これは MUGAL の一つであり, 標準 MUMPS 装備のシステムをもつユーザー施設に MUMPS ユーザーズ グループから適当な手続きののち配布される.

謝 辞

MUMPS 入門書の第1版は Martin E. Johnson と Ruth E. Dayhoff によって書かれた. この第2版は相当量の改訂部分をもつが第1版からの多くの文脈をそのままに残している. これらの初版の作業に著者らは謝意を表する.

この改版の草案は MUMPS 共同体の多くの人々にあらかじめ読まれ最終文章にもり込まれた多くの有意義な示唆をいただくことができた. このうちの主な寄与は Martin Johnson, Richard Zapolin, Henry Heffernan, Stephen Johnson および Arden Forrey の諸氏によるものであったのでここに記して著者らの謝意を表し, これら諸氏の助言にもかかわらず, なおも正確さを欠く文脈の責任は著者らにあることを表明する.

参考文献

1) Greens, R. A., A. N. Pappalardo, C. W. Marble and G. O. Barnett (1969), "Design and Implementation of a Clinical Data Management System," Computers in Biological Research, V. 2, pp. 469-485.

2) Munnecks, T., R. F. Walters, J. Bowie, C. B. Lazarus and D. A. Bridger (1977), "MUMPS: Characteristics and Comparisons with Other Programming Systems," Medical Informatics, V. 2, pp. 173-196.

3) Beckley, R. E, and D. A. Bridger (1977), "Advanced MUMPS Techniques," MUMPS Users' Group.

4) Brigham, C. R., J. D. Halvorson and J. Zimmerman (1976), "QUEST: A Teaching Program Driver," Journal of Computer-Based Instruction, V. 3, No. 2, pp. 42-50.

第 2 章

MUMPS の基本要素

文字セット

MUMPS は USA Srandard Code for Information Interchange (ASCII) に定められ，American Natioral Srandard 文書 X3. 4-1968 に記載された 7 ビットの文字コードを用いる．この文字セットは表 2·1 でみるように 26 の A から Z までの大文字，26 の a から z までの小文字，10 の 0 から 9 までの数字，および 33 の特殊文字をもつ，これらのほかに遠隔制御やそのほかの作用のための 33 の印字されない制御文字がある．プログラミングにはテレタイプに一般にみられる ASCII コード中の 64 の文字サブセットのみでおこなうことができる．それゆえに MUMPS を用いるために小文字の印字機能を必要とはしていない．

数(number)

MUMPS は数を含む数学的コンピューティングに用いることができる．数値のもっとも単純な形は正の整数の形であり，17，245，1967，や 100500789 は MUMPS システムで用いる整数の例である．整数は小数点以下の要素をもたないので 13.0 というのは小数点以下が 0 でも整数ではない．整数は正または負の記号をともなっていてもいなくてもよい．

多くのコンピュータ言語は整数と真数とを区別している．真数とは整数部と小数点以下の部分を含むもの，たとえば 2.3，7.5 または 100.500789 のようなものとして定義される．真数は正確度を高めた計算に，割算に，また小数点以下の結果を出す演算に用いられる．MUMPS ではユーザーにある演算では整数と真数とを区別しておこなうことを可能にしており，あとで述べられる．

表 2・1 ASCII (JIS)

この文字表記は ANS X 3.4-1968 に規定されたもので，JIS はそれの拡張である．
られるものである (0~31 は非印字文字)

コード	文字	コード	文字	コード	文字	コード	文字
0	NUL	32	スペース	64	@	96	‘
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	−	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\ (JIS では¥)	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	消去文字

文字表

文字のコード値は $ ASCII 関数の値として，また $CHAR 関数の引数として用い

コード	文字	コード	文字	コード	文字	コード	文字
128	NUL	160	スペース	192	タ	224	
129	SOH	161	。	193	チ	225	
130	STX	162	「	194	ツ	226	未
131	ETX	163	」	195	テ	227	
132	EOT	164	、	196	ト	228	
133	ENQ	165	・	197	ナ	229	
134	ACK	166	ヲ	198	ニ	230	
135	BEL	167	ァ	199	ヌ	231	
136	BS	168	ィ	200	ネ	232	
137	HT	169	ゥ	201	ノ	233	
138	LF	170	ェ	202	ハ	234	
139	VT	171	ォ	203	ヒ	235	定
140	FF	172	ャ	204	フ	236	
141	CR	173	ュ	205	ヘ	237	
142	SO	174	ョ	206	ホ	238	
143	SI	175	ッ	207	マ	239	
144	DLE	176	―	208	ミ	240	
145	DC1	177	ア	209	ム	241	
146	DC2	178	イ	210	メ	242	
147	DC3	179	ウ	211	モ	243	
148	DC4	180	エ	212	ヤ	244	義
149	NAK	181	オ	213	ユ	245	
150	SYN	182	カ	214	ヨ	246	
151	ETB	183	キ	215	ラ	247	
152	CAN	184	ク	216	リ	248	
153	EM	185	ケ	217	ル	249	
154	SUB	186	コ	218	レ	250	
155	ESC	187	サ	219	ロ	251	
156	FS	188	シ	220	ワ	252	
157	GS	189	ス	221	ン	253	
158	RS	190	セ	222	・	254	
159	US	191	ソ	223	・	255	消去文字

この区別をつけるのはプログラム内でユーザーがおこない，結果に及ぼされる影響に対する責任をとらねばならない．

ほとんどのコンピュータは2進法を用いるので小数点以下のもっとも意義の小さい要素のところでは正確さを欠くことがある．またコンピュータの小数点の扱い方にも違いがあり最後の桁の印字で表示されるものに差がでることがある．これらのコンピュータ間のわずかな差異は多くの問題での計算上の重要な正確さに大きく影響をするものでないが，このようなことが起こることは知っておくのがよい．

人間は長い桁数からなる数字列を読むことに困難を感ずる．慣習上出版物では大きい数字は普通たとえば 100 万のことを 1,000,000 とコンマで区切る．MUMPS ではこのような表示法は受付けないし，他のほとんどのプログラミング言語でも受付けない．そのかわり科学的計算法が用いられる．特別の表示法が採用されている．指数表示法というこの表示法はEという文字に後付けされ10 の何乗かであらわされる．次に若干の例を示しておこう．

通常の表示法	指数表示法
125	12.5 E1 (12.5×10(1))
1389.468	13.89468 E2(13.89468×10(2))
1000000	10. E5 または 10.0 E5
172.4	17.24 E1
0.0156	15.6 E−3
0.000000762	76.2 E−8

この表示法は非常に大きい数字や非常に小さい数字を表示するのに有用であるので科学的なコンピューティングで一般に用いられる．

数は一般に文字セットの特別な場合として考えることができる．この入門書では上記の例を numlit として参照することにする．numlit とは数値をもつものとして解釈できる一群の文字ということである．次にあげるものは有効な numlit とそうでないものの例である．

有効な numlit	無効な numlit
35	thirty-five
29000	29,000
10E5	10×10(5)
0.34	34/100
13.0E4	13.0　E4（スペースが入っている）
10	E+1　　（数値がない）

数値の範囲と精度(numeric value range and precision)

小数点以下の数値の表現は常に有限であり，３分の１の10進表現のように無限の表示ができるかというとそうではない．コンピュータ言語でも計算される数値の正確度について一定の精度を選んでおく必要がある．標準 MUMPS では整数でも真数でも９桁の精度を要求している．したがって 123456789 や 1.23456789 は両方とも有効であるが，99.87654321 のような数値は，99.87654320 として最後の意義をもつ数字が 0 として四捨五入されたまま格納されることになる．この標準 MUMPS 仕様規準は最小限９桁までの精度をもつこととなっているので，ある種の標準 MUMPS 装備では事実もっと大きい有効桁数を用意している．読者で９桁以上の有効桁数を必要とする仕事に従事する者があれば，自分の用いる MUMPS 装備についてベンダーにあらかじめたずねておくのがよい（さらに２，３の章まで進んでから精度を自分で検定してもよい）．

指数表示は一般の表示法よりもずっと大きいまたは小さい数値を表現することができるので，標準 MUMPS で認識されうる指数の大きさの限度慣習も採用されねばならない．ANSI 標準では 1E−25 から 1E25 までが，どの標準装備にも受け入れられなければならないとされる．ここでも多くの装備がこの最低規準の示す範囲よりも大きい範囲の指数表示を許している．これらの指数範囲でも仮数部の範囲は最低９桁が有効でなければならない．

定文字列 (string literals)

コンピュータ利用の中には自然語——名前，住所，文章体など——をソートしたり検出したりする仕事が多く含まれている．MUMPS はことさらこのような文字列 (string) を操作する仕事に適している．一般に，定文字列とは表2・1 に出ている ASCII 文字列のどのような組合わせによってもよい一連の単位をいう．だから前の節で示したすべての数値もそれぞれ文字列とみなすことができる．定文字列を numlit と区別するためには strlit と書くことにする．さきに述べたように，逆に定文字列は必ずしも numlit ではない．

strlit を定義するための慣習はこれらを引用符号（"）で囲むことになっている．それゆえ "ABC" は A, B, C という 3 つの文字をもつ strlit となる．数字，アルファベットその他印字されるすべての記号もこの定文字列に含むことができる．引用符号で定文字列が囲まれる慣習は引用符号そのものを定文字列に含もうとするときに支障が生じる．言語定義委員会はこれを解決するために定文字列内の一対の引用符号はそれぞれ 2 個ずつ書くことと，引用符号を間に文字や符号やスペースをはさまずに 2 個続けて書くときは Null（空）文字列として扱うことを決めている．実際の strlit 例から以上の定義をみてみよう．

定　義	実際の定文字列
""	空の文字列
"XYZ"	XYZ
"John Doe"	John Doe
"a""super""party"	a "super" party

文字列も数値と同じように無制限に長いものを扱うことはコンピュータの容量による制限があるので，ここでもその長さの最低限を ANSI 標準が決めている．どのような単一文字列も 255 文字を超えてはならないことと，標準 MUMPS の装備はここまでの長さの文字列を扱わねばならないことである．ある種の装備ではこの制限以上に文字列を受け入れる拡張をしているが，文字列の長さよりももっと必要なことは数値における精度を上げることなので，ほ

とんどの MUMPS 装備はこの文字列長の標準的最小限に従っている．

変　数(variables)

代数ではXなどの文字を等式の中で未知数の代用として用いる．X＝13＋22 ではXの値を問うことになるが，1つの変数を異なった場合にとる異なった値の代用として用いるのは便利なことで，コンピュータ言語ではこのような変数を種々の方法で定義する．MUMPS では変数は第1文字が大文字アルファベットか“%”記号であって，残りの文字は大文字アルファベットか数字であれば何を用いてもよいことになっている．これに合致する変数名としては X, NAME, %HGHT, R2D2 などが例示できる．z とか 25th, A＊B などは上記の定義に従わないので変数名として無効である．名前に小文字を用いることができるものがあるかもしれないが，この方法は多くの端末で小文字のアルファベットの入出力のできない現在では用いないようにすすめられている．

変数名を意識的に長くすることはできるが，一般には極端に長い変数名を用いて，データの操作を多くおこなうにはプログラミングが手間どることになる．標準 MUMPS では8字長までの名前 (name) が識別され，最初の8文字を超える文字は無視されることになっている．標準 MUMPS の最小限規定よりも長い名前を識別できる装備もあるので8文字以上の名前をつくらない場合にはまずこの点を確かめておくとよい．

FORTRAN など多くの言語では整数値，真数値，その他の値をとる変数名をはじめから識別させる規則をもたせている．MUMPS では変数を格納する際のこの種の区別は何も定めていない．これは MUMPS を他の多くのプログラミング言語と区別するのに重要な点である．つまり strlit 内の数字は，後述する言語仕様によって一時計算用に転用され，再び文字列におきかえられてのちにおこなう出力や格納にそなえられることになる．実際上の応用法についてはあとの章で解説される．

関　数(functions)

以上では MUMPS 言語の基準となるもの——数字，文字列，および変数名について述べたが，これらの基本的要素を操作する特殊な機能によって MUMPS も含むコンピュータ言語の力は左右される．このような操作をおこなうものの一種に関数というものがあり，それらは常にドル記号（$）ではじまる関数名で識別される．この入門書で詳しく解説される関数には $FIND, $PIECE, $SELECT などがある．ここではこういう関数があることと，それらが文字列や数値を扱って種々の操作をする言語要素であることを知るだけにとどめておけばよい．

特殊変数(special variables)

この章で最後に述べる言語要素は特殊変数である．関数のようにこれらもドル記号（$）ではじまる名前をもつ．関数と違うのは，これらの要素は単一数値，たとえば日付とか，時間（$HOROLOG），テキストの行の中でのカーソルの位置（$X）などをもつ変数である．これらについても詳しく後述する．

要　約

この章では，MUMPS 言語の基本的な要素——数値，文字列，変数，関数——について概要を知るにとどめた．変数には数値や文字要素が含まれることも知っておくとよい．これらの変数の値をどのように，おのおのの関数や他の言語要素が操作して複雑な問題の処理をするかについて，これから学ぶことになる．

第 3 章

数値の演算

第2章で MUMPS 言語ではデータには基本的に2種あって，一つは数値，他は文字列であることを学んだ．また，これらの2種とも時を違えて同一の変数名の中に格納されることも学んだ．この章ではまず数値を操作する演算子について知り，のちに数値にも文字列にも作用する演算について学ぶことにする．MUMPS で用いる概念のほとんどは基礎的な算術法であるから，容易に理解できるはずである．

二項演算子(binary operators)

MUMPS には数値を種々の方法で結合するために6個の演算子がある．それらは，

＋	加算
－	減算
*	乗算
／	除算
＼	整数除算
#	モジュロ

はじめの4個の演算子は四則演算として日常用いるものである．もっとも普通の形式では2個の数値に作用して第3の数値をつくり出すための二項性をもっている．

4＋7＝11

18－6＝12

3－4＝－1

9＊6＝54

10／2＝5

4／3＝1.33333333

最後の例は MUMPS は9桁の有効な桁をもつという点を再度強調している．4/3 の商は無限小数点 1.3333…… をもつが，さきに述べたように MUMPS は9桁以下を四捨五入している．残り2個の演算子は異なった形式の徐算である．整数商徐算は名の示すとおり整数値のみを返す．厳密にいえば演算子＼は通常の徐算（／）で計算された商の値の整数部分のみを返す．例で示すと次のようになる（最後の欄はあとで説明する）．

N	D	N/D	N\D	N#D
10	2	5	5	0
−8	4	−2	−2	0
6	−3	−2	−2	0
6	4	1.5	1	2
1	3	0.33333333	0	1
−14	−6	2.33333333	2	−2
4	−3	−1.33333333	−1	−2
13	5	2.6	2	3

モジュロまたは剰余徐算は整数徐算の補足的な演算で＃演算子によって返される値は整数徐算で取られた部分の残りの部分である．正の数値に対しては，よく知られている割り切れない残りとなる．たとえば 13 を5で割ると商の3が立ち残りは3となる（上記の最後の例）．モジュロの数学的な式は次のようになる．

N＃D＝N−(D＊floor(N/D))

ここで N は分子で D は分母であり floor(X) とは X に等しいか X を超えない最大の整数値として定義される．上記の最後の2例の数値を当てはめてみると次のようになる．

4−(−3＊floor(4/−3))

4−(−3＊floor(−1.33333333))

4−(−3∗−2)

4−(6)

=−2

また,

13−(5∗floor(13/5))

13−(5∗floor(2.6))

13−(5∗2)

13−10

=3

13−((13\5∗5)

13−(2∗5)

13−10

=3

以上の定義を知って上記の例のすべてのモジュロ値を検算してみるとよい．このとき結果の正負記号は必ず分母のそれと一致することを知っておくとよい．

算術式(arithmetic expressions)

前述した算術演算子 (arithmetic operators) は種々の方法で算術式を構成するのに用いられ，算術演算子と値とを組み合わせた算術式は1つの値を生ずる．前節で簡単な式を示したがそこでは2つの値が1つの演算子をはさんでいた．しかし式にはもっと多くの要素が混合することがある．たとえば,

1+2+3+4+5+6=21

は数個の数値と演算子の組合せによる式であるが，ただ1つの値を生ずる．もっと複雑な式となって多種の演算子を含むとき MUMPS のようなコンピュータ言語では一定の値を出すための規則がなくてはならない．3+5∗2 という式を考えた場合での演算子を先に評価するかで可能な2種の解釈が生ずる．どちらか1つの解釈をさせるには括弧を用いて (3+5)∗2 として 16 を得るか,

3+(5*2) として 13 という結果を生じさせるかが考えられる．MUMPS では他のコンピュータ言語と同じように演算の順序を制御するためこのような括弧を用いてもよい．しかし，もし括弧のない場合 MUMPS はどう解釈するであろうか．MUMPS は括弧のない式を単純に左から右に出会った順序で評価していく（この左から右への優先順位はある種のコンピュータ言語では用いないことに注意）．このようにして今あげた括弧のない算術式は MUMPS 言語では (3+5)*2 と書かれたかのように扱われ，16 と評価される．当然，括弧を用いればその内の式が最初に評価されるので上の順序を逆にしたい場合には 3+(5*2) と書かねばならない．MUMPS がこの式に出会えば括弧内の値を評価してから，13 という結果を出す．

左から右への評価は他のコンピュータ言語を知らないプログラマには自然にみえ，何の問題もないが，他のたとえば FORTRAN のような言語に経験のあるプログラマははじめ MUMPS の定義する優先順位にとまどいを感ずるであろう．FORTRAN の場合には加算や減算に乗算と徐算が優先して評価されるからである．このようなときには括弧を用いて解決していくのがよい．この場合計算順序の点で不安は生じなくなる．

最後に１つの例をあげて，複雑な式がどのような算術優先順位 (arithmetic precedence) をもつかをみておこう．各行で下線をつけたところが最初に評価される部分である．

17−4/2(8*3)\6#2

13/2+(8*3)\6#2

6.5+(8*3)\6#2

6.5+24\6#2

30.5\6#2

5#2

1（最終値）

前掲のすべての例では，数値としての定数を用いてわかりやすくしてきた．MUMPS は算術式の中に変数を用いることができる．たとえば次のような数

値を変数に割り当てておくとする．

A=17　B=4　C=2　Y1=3　BCX=6　X=8

そして前例は次のように書くことができよう．

A−B/C+(X＊Y1)\BCX#C

これは全く数値で示したと同じ順序で評価されて，同一の結果を生じる．どんな算術式も定数と変数と算術演算子との組み合わせと，評価に必要な括弧をほどこしてつくることができる．

一項演算子(unary operators)

算術演算子のうちの2個（＋と－）は演算項目の前に単一的に用いられることがある．この場合2つの演算項目を結ぶ二項演算子ではなく一項演算子として扱われる．これらは特に算術用の変数を扱うときに用いられ，そのデータを文字列から数値に変換して必要に供する．一項のプラス記号は文字列をその数としての等価値に変換する．マイナス記号はこの変換値の記号を逆にする．たとえば，A=0，B=3，C=−6という3種の変数値があるとき，これらの一項演算子で変換される結果は次のようになる．

+A=0

−A=0

+B=3

−B=−3

+C=−6

−C=6

数値解釈(numeric interpretation)

一つの例外のみを除き，以上で算術式の説明は完全であるが，ここで想起してほしいのは変数には数値および文字列値が入りうることである．A+3という式ではAは数値をもつことを加算の定義が強制している．この事実についての心配はいらない．それは MUMPS が自動モード変換という機能をもつから

で，それはAがもし A+3 を評価する時点で文字列値をもっておれば，MUMPS がAを算術式を成立させるように変換してくれるということである．Aが文字列定数として“1 2 3”を入れていても，A+3 の結果は 126 となる．MUMPS が文字列を数値に変換するときの規則をここで知っておく必要はない．単にここでは単純なルールを，数値の定義でみたときのことを基にして知っておくだけでよい．文字列の左端に有効に定数として用いうる部分があれば（もし一項演算子がついていればそれを除いたあと），この文字列はその定数の数値として変換され，もし有効な定数となるものがないときは MUMPS はそれに 0 という値を割り当てる．理解を助けるために若干の例をあげる．

文字列	数値
“ABC”	0
“”（空文字列）	0
“123”	123
“−12”	−12
“+36.5”	36.5
“12 ABC”	12
“3.1 E−3”	.0031
“A12 B36C”	0
“+−0.362”	−0.362
“9E 2AB”	900

一項演算子のもっとも一般的な用い方はそれのつくオペランドを数値として強制解釈させるためにおこなわれる場合である． MUMPS では数値が絶対的に必要とされる場合がよくあるが，このときに，一項演算子の + を用いると MUMPS は数値解釈をおこなう．上の例で文字列ないしこれらをもっている変数名の前に+を置けば対応する右側の数値を生じる．

算術関数—$RANDOM

標準 MUMPS では，ただ一つ定義されている算術関数（arithmetic func-

tion）がありそれは $RANDOM である．この関数は0に等しいか1より大きい整数1個だけの引数をとり，この関数が実行されるごとに0から（引数−1）の間の整の偽乱数を生じる．たとえば，$RANDOM (100) とすると0から99までの間の，$RANDOM(10) では0から9までの間の整の偽乱数を $RANDOM(1) とすれば0のみを（したがってこの場合意味がないが）生じる．この関数は乱数値の必要な算術計算に有用となる．

要　約

この章では単純な例として算術演算子を用いて関数と演算子の概念を紹介した．これに次いで，第4，5章では他の2種の演算子である論理（または比較）演算子および文字列操作の演算子について解説する．

第 4 章

論理演算子（比較）

前章では算術演算子が変数に値を設定したり，定数や変数を用いて計算したりすることを学んだ．コンピュータの応用プログラムでは２つの変数を比較して，その結果によって別の行動をとらせることが頻繁にあり，すべてのコンピュータ言語には何らかの形でこの種の比較検定をおこなう機能がある．MUMPS はこのような処理のためにいくつかの比較操作の演算子をもっている．これらの演算子についてこの章で解説する．

算術演算子と関係演算子(relational operators)

MUMPS では変数値を文字列として格納しているので数値は数種の方法で変数の中では表現されている．これらを数字列としてみるか文字列としてみるかの区別がつかなければならない．35, 3.5E1, .35E2, などは同じ数値であるが格納のされ方においては異なっている．数値として比較するための２つの演算子がここにある．

＜　より小さい (less than)

＞　より大きい (greater than)

これらの演算子が間に入ると変数値はまず数値として変換され，次いで比較がおこなわれる．

上の２つの算術関係演算子 (arithmetic relational operators) に加えてなお３つの文字列関係演算子 (string relational oprators) が用いられる．
それらは，

＝　等しい　(equals)

[　含む　(contains)

]　追従する (follows)

2つの文字列が等しいかどうかを検定するはじめの演算子は，その比較が数値上のものでなく，これを用いた場合 35 と 3.5E1 は等しいとして評価されないことに注意すべきである．ここで数値上の等しさを検定しようとするならば，一項演算子＋を用いて強制数値解釈をさせることになる．かりに，

A＝“35”　　B＝“3.5E1”

とすれば，

A＝B　　は真でなく，

＋A＝＋B　　は真である．

これは数値解釈が強制されたからである．プログラマは数値が問題になるときこの二項演算子（＝）を用いるには注意を必要とする．

それ以前の数値操作によって数値が生成されているとき，たとえば加算によって生成された文字列のような場合には，これらの文字列は MUMPS の“標準数値”形式になっている．これらの数の“値”についてはただ一つの形式しか存在しないから，直接＝演算子を用いてもよい．B＝“33”＋“2” を実行し，そして B＝＋“3.5E1” を実行したのちは，

A＝B

は真となる．

包含 (contains) 演算子（[）は1つの値の中に他の値が含まれるかを検定する．今，VAR に “ABC 123” が割り当てられ TEST1 と TEST2 に “BC1” と “C2” がそれぞれ割り当てられているとして，次に若干の検定例とその結果を示す．

VAR [“A”　　真と評価される（VAR 内に A がある）
VAR [TEST1　真と評価される（VAR 内に BC1 がある）
VAR [TEST2　偽と評価される（“C” も “2” も VAR 内にあるが，これらは一度に起こっていない）

1つの変数が他の変数に追従 (follows) するというのは，ASCII 表の順によって検定するのである．ASCII－JIS 表は表 2・1 に示されている．上記の3つの変数を用い演算子] で評価すれば，

VAR] TEST1 偽と評価される

TEST1] TEST2 偽と評価される

TEST2] VAR 真と評価される

この演算子は特に名前やその他の文字列をアルファベット～アイウエオ順で並べかえをするのに有用である．それは ASCII-JIS 順位はアルファベット～アイウエオ順をそのまま用いているからである．比較は１字ずつ左から右に向かっておこなわれる．

以上の例で真（true）であるとか偽（false）であるという評価をするとき，それを表現する MUMPS の出力は真のときに 1，偽のときに 0 の値である．これらの値 0 と 1 を真偽値（truth value）といい，真偽値を生ずる評価の対象となる式を真偽値式（truth value expression）という．

論理演算子 (logical operators)

前章とこの章とで解説した算術演算子と関係演算子のほかに，MUMPS では論理演算子を用いて条件の結合を検定することを可能にしている．

&	アンド	(and)
!	オア	(or)
'	否定	(not)

アンド (and) 検定ではその両側の条件が真であるときのみに真の結果となる．オア検定ではその両側の条件のどちらかが真であれば，真の結果となる．ここで X が "ABC" という値をもち Y が "DEF" という値をもつとするとき，論理演算子を混じえて以前の式の検定をおこなうことができる．

"ABC"=X	真	(1)
"AB"=X	偽	(0)
X ["AB"	真	(1)
"AB" [X	偽	(0)
X [Y! (X["AB")	真	(1)
Y] X & (Y["EF")	真	(1)

X] Y & (X ["B") 偽 (0)

12 > 11 ! (X=Y) 真 (1)

上の最後の4例では左から右への順で正しく評価されるように括弧のついていることに注意.

否定(not)演算子(')はそれの付けられた式の真偽値を逆転させるために用いられる. それで, '(11>12)は真として, '("ABC"="ABC")は偽として(本来の真偽値を逆転して)評価される. 否定演算子はまた関係演算子や論理演算子と結合して用いることもできる. たとえば,

11'>12 は真であり, また

"ABC"'="ABC" は偽である

一般に A'関係演算子 B は '(A 関係演算子 B) としても同じことである.

上記の演算子を使えば, 複雑な関係の組合わせを考えてみることができるが, 結果としてはただ一つの真または偽の値が生ずるだけである.

要 約

この章では, MUMPS 関係演算子の紹介をおこなってきた. 前章で述べた算術演算子とともに使うことにより, これらは一組の強力な道具となり, プログラマは数や論理的関係を含む多くの仕事をすることができる.

次の章では, ユーザーが文字列を操作することを可能にする演算子の最後のグループについて解説する.

第3, 4, 5章では, MUMPS 言語操作の主要なツールを学んでしまう. MUMPS は特に文字の扱いを可能にするよう考案されており, MUMPS が他の多くのプログラミング言語と異なる性格はこのシリーズの最後の第5章で述べてある.

第　5　章

文字列の演算

はじめに

第３，４章で，まず算術演算子と関数を，次いで論理演算子と関係演算子について学んだ．この章では文字列操作上用いられる演算子と関数の最後のグループについて解説する．これらの言語ツールを習得すれば，さしずめプログラマが利用する語彙は完全になる．この章ではパターン照合演算子（第13章で別に解説）以外の文字列演算子について学ぶ．

文字列や文章は多くの方法で操作することができる．文字列の一部（部分文字列，substring）は，その位置を確かめたり，分離したり，他の文字列へ移したりすることが可能である．ある文字列を他の文字列と結合することもある．第４章で述べた関係演算子を使うことにより，文字列をその長さや，ある特定の文字の存在に関して検査したり，比較したりすることもできる．MUMPSは広範囲の文章操作も許されるように考慮されているため，これらの機能はこの章で述べる関数や演算子を用いて容易に達成できる．これらの概念は実際の応用例の中でのみ完全に理解できることであろう．読者が以下に示すMUMPSの能力を自分で工夫した問題についても実験してみることをすすめたい．

文字列演算子(string operators)

第４章で，文字列に作用する３個の演算子を示した．それらは，

演算子	作　用
A＝B	AとBについて等しいかどうかを比較する
A [B	AがBを含んでいるかを判定する
A] B	A が ASCII(JIS) 配列順位においてBに追従するか

を判定する

これらの演算子グループに，ここで第4の演算子をつけ加えてみよう．それは_（アンダーライン）であらわされる．これの機能は1つの文字列を他の文字列のうしろに付加すること，あるいは文字列同士を結合することを目的とする．以下に示す例は定文字列と変数を用いた用法である．ここで A="ABC", B="MUMPS" と設定されているとする．

評価される式	結果の値
A_"DEF"	ABCDEF
A_B	ABCMUMPS
A_"'S OF "_B	ABC'S OF MUMPS

この演算子は，特に文字列型変数を挿入して，メッセージをつくるときに有力である．たとえば，USER という1つの変数に，端末装置を利用する者の名前を設定しておけば，次のような文字列ができるであろう．

"HELLO, "_USER_"! WELCOME TO MUMPS. "

結果として書き出される文字列は，利用者の名前を間に挿入した，3個の文字列がいっしょに結合された一連の文章となっているだろう．

文字列関数(string manipulating functions)

第2章では，$RANDOM 関数と，その算術演算での利用法を学んだ．これから，効果的な文字列操作に必要な手段を提供する関数について検討を加える．

$LENGTH 関数

文字列の一つの重要な性質は，その長さである．それは，文字列がある与えられた区切りの中に含む文字の数である．$LENGTH はどんな文字列の長さをも決定して 1 つの数値を返してくる．その数値は最小は 0 であり，最大は MUMPS が許している文字列の最大限である 255 までが可能である．以下に，いくつかの例とその働きを示す．はじめに，T="JOE BROWN"，V=

"ABC : DEF : GHI", DAT="9/5/55", X="TOWN" と設定しておけば，

評価される式	結果の値
$LENGTH ("") (空文字列)[訳注]	0
$LENGTH (T)	9
$LENGTH (V)	11
$LENGTH (DAT)	6
$LENGTH (T_X)	13
$LENGTH ("THE END")	7

ここで，($LENGTH (T)+$LENGTH (X)) と $LENGTH (T_X) とは同じ値をとることに注意すべきである．いいかえれば $LENGTH 関数は１つの文字列や変数に対してというのでなく，１つの式に対して評価をおこなうことができるのである．

$FIND 関数

$FIND 関数は，ある部分文字列が，他の文字列内にあるか，あるならどの位置にあるかを求める関数である．この関数はある文字列内の特定データの位置がわからないときに，しばしば用いられる．$FIND の第２引数としては第１引数の文字列の特定の部分や文字列全体を扱うことができる．$FIND を実行すると，第１引数で指定された文字列内にある第２引数の位置を返す．返してくる結果の値は１個の整数値であり，第１引数内で第２引数が生起する位置のすぐあとの文字の位置を示している．第２引数の部分文字列が，第１引数の文字列内にあらわれないときは $FIND の値は常に０（ゼロ）となる．

ここで２個の変数を，S="ABABCDEFABCXYZ"，および SS="ABC" と設定すると，

評価される式	結果の値
$LENGTH (S)	14

訳注）定文列としてのデータではあるが，長さも文字ももたないものを空文字列 (empty string) といい，通常 CR でこれを代用する．

$FIND (S, "AB")	3
$FIND (S, SS)	6
$FIND (S, "PQ")	0

$FIND はまた第3引数をとることができる．第3引数は第1引数の文字列内で第2引数の部分文字列を探しはじめるべき文字の位置を指定する．したがって上掲の例と同じ2個の変数を用いれば，

評価される式	結果の値
$FIND (S, "AB", 2)	5
$FIND (S, "AB", 4)	11
$FIND (S, SS, 4)	12
$FIND (S, SS, 10)	0

$EXTRACT 関数

$EXTRACT 関数はおもに複合文字列の中からある定められた長さの部分文字列を抽出されるために用いられる．$EXTRACT 関数は $FIND 関数の場合と同様に，2個ないし3個の引数をとることができる．2引数型の $EXTRACT は，第2引数によって指定された文字の位置にある1個の文字を第1引数から取り出すために用いられる．たとえば，S="ABCDEFGHIJ" であるとすれば，

評価される式	結果の値
$EXTRACT (S, 5)	E
$EXTRACT (S, 7)	G
$EXTRACT (S,$LENGTH(S))	J
$EXTRACT (S, 0)	"" (空文字列)
$EXTRACT (S, 11)	"" (空文字列)

3引数型の $EXTRACT では，第2，第3引数は抽出したい部分文字列の，それぞれはじめと終りの文字の位置を指定している．前掲の例の変数値を用いると，

評価される式	結果の値
$EXTRACT (S, 2, 5)	BCDE
$EXTRACT (S, 6, $LENGTH(S))	FGHIJ
$EXTRACT (S, 7, 7)	G
$EXTRACT (S, 8, 7)	“” (空文字列)

最後の例では，第3引数が第2引数よりも小さいので，部分文字列は何も指定できず，結果は空文字列（“”）となる．

$PIECE 関数

$PIECE 関数は，1個あるいはそれ以上の境界を定める文字を入れた文字列から部分文字列を分離するために，さらに強力な手段を提供する．$EXTRACT 関数ではひき出す文字列のはじめや終りの文字の位置を指定してやる必要があったのに比して，この $PIECE 関数は，その文字列内の位置に関係なく与えられた1個の境界子（delimiter，境界を指定する文字）を頼りに文字列を抜き出してくれる．

$PIECE を用いるとき，抜き出す部分文字列の場所を検索させるために知っておくべきことは，それを含む文字列内で抜き出すものが，他の部分文字列に対してとる相対的位置と，これらの部分文字列を分けるために使っている境界子文字の2つである．$PIECE の第1引数は部分文字列をとり出すもとの文字列をあらわし，第2引数は境界子の文字であり，第3引数は求める部分文字列の相対的位置を示している．例を示すために，ここで SD=“LASTNAME, FIRSTNAME” および S=“ABC/DEF/GHI” と定めて，これらの2つの変数と2個の異なった境界子を用いてみる．

評価される式	結果の値
$PIECE (SD, “, ”, 2)	FIRSTNAME
$PIECE (S, “/”, 3)	GHI
$PIECE (S, “/”, 1)	ABC
$PIECE (S, “/”, 4)	“” (空文字列)

$PIECE 関数では $EXTRACT 関数での場合と同様に，取り出す部分文字列の範囲を指定するための第3引数を用いることができる．次に示す列は前掲の例と同じ変数とそれらの値を用いて，この型の $PIECE を解説している．

評価される式	結果の値
$PIECE (S, "/", 1, 2)	ABC/DEF
$PIECE (SD, ", ", 2, 5)	FIRSTNAME

境界子の文字は単なる句読点のみに限らない．X="A" として次の例を考えてみる．

評価される式	結果の値
$PIECE (S, "DE", 1)	ABC/
$PIECE (SD, X, 2, 3)	STNAME, FIRSTN

この関数を用いると，相当長いデータ文字列から部分文字列を検索することが比較的容易になり，MUMPS プログラマは将来分解して用いるつもりで，変数を連結してそれらの値を圧縮した形で格納することができるようになる．事実，$PIECE は MUMPS 言語の中でもっとも有用な文字列関数の1つであり，他のコンピュータ言語のほとんどのものでこれと同じことをおこなおうとすると，特殊な複雑なプログラミングが必要となるであろう．$PIECE 関数の応用例はこの入門書の後半の章で示されるはずである．

$ASCII 関数

ときとして，ある文字列内の特定の文字の ASCII (JIS) コード番号を知ることが必要になることがある．$ASCII 関数は2個の引数を用いてこれを可能とする．第1引数は調査対象となる文字列を指定し，第2引数は ASCII コード値 (JIS コード値) を求める文字の位置を指定している．もし第2引数が省略されておれば，文字列の最初の文字の ASCII コード値 (JIS コード値) が求められる．$ASCII によって返される番号は，0から 127 (JIS では0から 223 までと1個の 255) までの間の ASCII コード (JIS コード) の範囲内の10進数である．ここで2個の変数を XX="ABC123"，YY="Q：R！" と設

定すると，

評価される式	結果の値
$ASCII ("A")	65
$ASCII (XX)	65
$ASCII (XX, 4)	49
$ASCII (YY, 2)	58
$ASCII (YY, 4)	33
$ASCII ("")	−1

$ASCII では空文字ないし，そこに文字のないときは −1 が返されることに注意.

$CHAR 関数

$CHAR は，$ASCII 関数の逆を実行する．この関数は引数によって指定された値に対応する ASCII 文字表 (JIS 文字表) の配列順位の文字を返す．引数値にはエラーを避けるため，ASCII 表 (JIS 表)——表 2・1 参照——の配列順位以外の 10 進値を用いてはならない．負の引数値は無視される．複数の引数があるときは，それぞれの引数に対応する文字がその順序で 1 つの文字列として返される．

評価される式	結果の値
$CHAR (72, 73, 89, 65)	HIYA
$CHAR (77, 65, −40, 67, 75)	MACK
$CHAR (−10)	""

要　約

この章では文字列操作の際，通常に使用される演算子と関数をとりあげた．実際の応用法を例示するプログラムはあとの第 17，18 章にゆずることになる．それまでに読者は MUMPS で完全なプログラムを書くすべての言語要素を学習し終えているはずである．

第 6 章

命令語への入門

命令文とは何か？

次の３つの英語の文を考えてみる．

Stop.

Open the door.

Put the bcok on the table.

これらのどの文も，誰かに何かをさせるための命令文である．他の人に向かっていう場合，われわれはこのような指令をやわらげて「どうぞドアを開けてください」などというが，それは要求について少し蔭をつけ，ややあいまいにしている訳である．「止まれ！」というのに比較して「どうか止まって下さい」というのは，情況次第ではまったく別の意味になってしまう．

コンピュータに何かをやらせる場合できるだけ意味を明瞭にし，あいまいさをなくするのがよい．それで無作法であっても直接的ないいまわしで要求を告げることになる．有難いことに，コンピュータもこのような通信を他の形式と混同しないために好むのである．

前述の英語の文章はすべて明瞭な命令文であり，ただそこにドアや本やテーブルさえあればよい．第１文は１語の長さをもち，単独な行為そのものを命じている．第２文は行為を何かに対しておこなうことを命じている．第３文では行為を何かと別の何かに対しておこなうよう命じている．もっと複雑な命令を出すこともできるのであるが，ここでは以上の例にとどめておく．

コンピュータの命令文はいろいろな点で英語の直接命令に似ている．ある命令文は１語長で，たとえば HALT (この命令語については後述する) がある．ほかの命令文は行為を他の要素に対しておこなうためのもので，そのことを命

令語が引数 (argument) をもつという．命令語 (command word) は引数をもたないか，1つの引数だけをもつか，あるいは1つ以上の引数をもちえることになる．引数が2個以上あるときにはコンマで区切ればよい．

SET 命令

MUMPS の命令語 SET を例にとると，これは1つの変数に対して1つの値を割り付けるための命令である．SET X=3 とすればXという変数に3という値を割り付ける命令文となる．Xという変数が今までにない場合にはXという変数をつくって，それに3の値を入れよということになる．Xという変数がそれまでに存在しており，たとえば "ABC" という値を入れている場合には上の命令文を実行すると，それまでの "ABC" という値は消えて新しい値 (3) がそれに置きかえられることになる．MUMPS は変数に入りうる値について数値か，文字列かなどデータ形式の区別をつけないので，どんな変数に，いつ，どんな形式の値を割り付けることも可能である．SET NAME="JOHN DOE" とすれば NAME という変数に "JOHN DOE" という strlit が割り付けられることになる．第2章で解説した算術演算子を用いることもできる．

SET Y=X+7

という例では，もしXが3であれば，この命令文の実行後Yには 10 という値が割り付けられている．もしもXに何の値も割り付けられてないときには，この命令文は実行できないので，エラーというメッセージが MUMPS から出される．

MUMPS はこの SET 命令を数種の用い方ができるようにしている．以上で示した例のように単独の SET のほかに，一度に数個の変数に新しく値をつけるために用いることもできる．この場合を複数 SET といい，たとえば，

SET (X, Y, Z) =20

として，X, Y, Zの変数それぞれに 20 という値を割り付けることができ，同じように，

SET X=5, Y=2, Z=X*Y, W=X>Y

とすることもできる．この場合Ｘがすでに何かの値をもっていてもそれは５という値に置きかえられ，同様にＹには２があてられる．したがってＺにはこれらの乗算による 10 が割り当てられる．ＷはＸ＞Ｙの式の値が真であるから真偽値の１をとる．この例から，命令語が複数の引数をとるとき，引数は一時に１つずつ左から右に向って実行されることがわかったであろう．

KILL 命令

いったん SET 命令で変数が定義されると，変数名のリストからこの変数とその値を除いてしまうには KILL 命令を用いなければならない．SET 命令のように，KILL 命令にも種々の用法がある．単一の引数をとったときには，プログラマが見ることのできる変数名とそれらの値の表から１つの変数とその値とを消してしまう．２個以上の引数をとる場合にも，それぞれの変数とそれらの値を消してしまう．引数をとらない KILL は，そのときに存在するすべての変数とその値を消してしまう．すなわち，

実行されるコード	実行結果
KILL A	変数Ａとその値が消される
KILL B, C	変数ＢとＣが値とともに消える
KILL	全部の変数が値とともに消える

以上で述べた KILL を用いる際には，消去すべき変数を指定するのに括弧を用いていないことに注意すべきである．括弧を用いるもう一つの KILL の用法があり，それによって KILL の実行後どの変数が保存されるべきかを示す．これは「(...) 以外をすべて KILL せよ.」というのと同じである．たとえば，

実行されるコード	実行結果
KILL (A)	変数Ａ以外のすべての変数が消される
KILL (B, C)	ＢとＣ以外のすべての変数が消される

上のあとの例は多引数の並記形式（括弧の外部で並記するもの）ではないから，したがって KILL (B), (C) とは同じものではない．

$SELECT 関数

例をかえて，$SELECT 関数をみてみよう．$SELECT 関数は真偽値式と式がコロンで分けられペアとなった複数の引数の中から真の値をとる単一の真偽値式を選び出す．これを例示するため，次のような変数の値が設定されているとする：A=1，B=2，C=3．新しい変数 RESULT にその値を設定しようとして，そのようなコードを書くことができる．

```
SET  RESULT=$SELECT(A>B:1, A=B:2, A<B:3)
```

MUMPS システムでは $SELECT 関数で括弧の中にあるコロンの左側の真偽値式を，左から右に向って真なる値を出すものに出合うまで探してゆく．この例ではAはBより小であるから，第3の真偽値式が真であり，その結果その真偽値式のコロンをはさんで右側の式，ここでは3が RESULT に割り当てられる．

$SELECT は並記してゆく真偽値：式のセットの末尾に，すべての真偽値式が偽であった場合の“欠損代用オプション”をおいて用いることができる．そのような構文を例示すれば：

```
SET  RESULT=$SELECT(A=B:4, A=C:3, 1:0)
```

この場合AはBにもCにも等しくなく，末尾のセットのコロンの左側の真偽値式は真 (1) であるので，変数 RESULT には0が設定される．$SELECT 関数の引数内には少なくとも1個の真の値をとる真偽値がなければならない．さもないとエラーとして実行されないことになる．コンピュータに命ぜられた仕事が実行できないとき MUMPS システムは「エラー」が起こったとして実行を停止し，この「エラー」は常にプログラマの失敗として（これは MUMPS のすぐれた方法である）みなされ，ただちにプログラマがそれとわかるような略式記号で知らせてくれることになる．エラー メッセージは，用いる MUMPS システムによって異なるが，コンピュータがわれわれプログラマに助け舟を出してくれ，われわれが注文していることができない誤りの原因を見付け出してくれることは有難いことである．

$SELECT 関数を使用する例の，もう 2，3 についてしらべてみよう．ここでもまた，A=1，B=2，C=3 であるとする．

実行されるコード	実行結果
SET RESULT=$SELECT(A>B:1, A>C:2, A<B:3)	3
SET RESULT=$SELECT(A=B:1, B=C:2, A=C:3)	エラー
SET RESULT=$SELECT(A=B:2*C, A=C:A+B, 1:0)	0
SET RESULT=$SELECT(A>B:A, A<B:C, A<C:B)	3 (C の値)
SET RESULT=$SELECT(A=C:"LB 1", A<C:"LB2", 1:"LB 3")	"LB 2"

上の第 4 の例では真の値をとる真偽値式が 2 個あることに注意．この場合 $SELECT は左から右に実行するときに最初に出会った真の値を返す真偽値式を選び出す．

WRITE 命令

以上でわれわれは 1 つの典型的な命令と 1 つの典型的な関数をしらべてみた．ここでさらに他の命令とそのオプションについてみてみよう．ユーザーの端末に向かってメッセージを出すための命令語 WRITE をみると，WRITE 命令は単に引数とする文字列を反復書き出しさせるのかもっとも単純な形で，WRITE "ジョージ ワシントン ハ ポトマック ヲ ワタッタゾ" とすればこのメッセージは端末にそのまま書き出される．何か計算した結果を入れている X の値をみたいときには，

WRITE "ケッカ ノ アタイ ハ:", X

X はあらかじめ計算されていなければならない．この際書き出されるものは " " 内のメッセージと今変数 X に入っている値である．この例で，メッセージは引用符号" " の中に入れておく必要があるのがわかるであろう．さもないとコンピュータは定文字列と変数名との間の区別ができなくなるからである．

プログラマが端末のスクリーンまたはページのどの位置に WRITE メッセ

ージを書き出せばよいかを指示するためには，数種の書式制御（format control)がある．制御文字“！”がそれ単独で引用符号“ ”なしで WRITE 文中にあるときには，それ以後のメッセージを次の行に出力することにある．ゆえに，

WRITE “コンニチワ”, !, “ワタシ ノ ナマエ ハ”, NAME

とかけば，出力は2行となり，第2行には ワタシ ノ ナマエ ハ とそれに次いで NAME に割り当てられた内容を書き出すことにある．ここで第1行目の書き出し欄は第0欄であり，その書き出しページのはじめの行は第0行ということになる．

同様に特殊文字“#”は書き出し用紙の新しい頭部または新たに CRT スクリーンの最上位から書きはじめることを意味する．また特殊文字“？”のあとに整数値をともなうときには，整数値で指定された，その行の欄から書き出すことを意味する．例として，

WRITE #!!, “トクイサキ”, ?10, NAME

これを実行すると CRT スクリーンはいったん消えて，カーソルが左上に動き（# による駆動），2行下って（!!による駆動），そこにトクイサキと書き，頭から第11欄目まで右に飛んでそこに NAME の内容を書き出すことになる．

WRITE 命令と $X および $Y 特殊変数[訳注)]

プログラマはときに NAME などの変数値がどれほど長い文字をもっているか知らないことがある．この場合，美しい出力のためには書き出す要素の位置を知りたいはずである．それによって意図する形に書き出しがおこなわれれば，出力の書式をほんとうに制御できたことになる．2つの特殊変数 $X と $Y は MUMPS がプログラマにこのような書き出し位置の情報を提供するために用意されている．$X は現在のカーソルまたは印字エレメントが行のどこにあるかを常に整数で格納している変数であり，$Y はスクリーンないしペー

訳注）すべて特殊変数は MUMPS が内部にもち刻々変化する値を入れているが，ユーザーがその値を直接 SET することはできない．

ジの上から何行目に現在あるかを変数として入れている．この2種の特殊変数がいつでも用いうるので，たとえ種々の変数内容が長さのまちまちのものであってもきれいな書き出しをできるようにしてくれる．

READ 命令

READ 命令の前に WRITE 命令を紹介したのは，READ は WRITE と同じように“メッセージ”を端末に書き出すことができることと同時に，これは，ユーザーからの端末入力を要求する点がやや異なっているからである．WRITE の場合は引数の変数が直ちに書き出されるが，READ の場合には要求される変数に値を入れるためにある時間話の緒口をつかんでいないとユーザーにとまどいを感じさせることになり，用い方としては READ も WRITE と同様な書き方が許されることをまず知っておけばよい．異なった点は READ では指定された変数の中にユーザーがそれに相当する内容を入力しなければならない要求をつきつけられているという点である．READ 命令は次の例のようにある種のさい促（prompt）をともなっているといえる．

READ !,“ネンレイ＝”,AGE,!,“セイネンガッピ
(MO/DA/YR)＝ ”,DOB

READ 命令の実行によって，これの並記引数に含まれる定文字列はユーザ端末に出力ディスプレイされる．出力書式文字 #，!，および？式は WRITE 命令文内に存在したごとく動作される．上記の例ではプリンター（または CRT）では1行改行され，第1番目のメッセージを出力し，そしてユーザーからの入力を待っている．その後次の行に移って次のメッセージを書き出したあと変数 DOB に対する入力を待つことになる．READ の場合メッセージの中に端末からのどんな入力でも受け入れて，それを DOB のように指定された変数に格納してしまう．ここで受け入れられる「値」というのは，装備が指定する終結子（ターミネーター，通常 CR）がくるまで，そして，その終結子を含まない文字列をとることになる．次に何らかの命令文があっても，この終結子が入力されるまでは実行に移らずに，そのまま待っている．

命令の後付け条件(post-conditionalized commands)

MUMPS の命令は，真偽値で評価されるある値ないし式によって条件付きで——真であろうと偽であろうと——実行させる能力をプログラマに与える．この操作を後付け条件法（post-conditional）という．

SET X=X+1

ここでは命令が実行されるたびに現在のXの値に1が加えられXは新しい値に置きかえられる．しかし，もしプログラマがXの値が 10 になったら，もうそれに加算をしたくないと考えたときにはどうすればよいだろう．この目的を達する一つの方法は元来の命令に後付け条件をつけておくことである．その方法でつくった新しい命令は次のようになる．

SET : X<10 X=X+1

ここで追加された文字群 :X<10 は後付け条件といい，コロン（:）は後付け条件のための式があることを示す．そして X<10 というのが後付け条件用の式そのものである．命令が実行されるとき MUMPS は後付け条件の存在を見分け，SET 命令をその後付け条件が偽として評価されると同時にその実行を中止する．以上の例では後付け条件のブール値が偽となったときに MUMPS は SET 命令を中止してXに加算をおこなわなくなる．ゆえにXは次にくる命令がはじまるときには 10 の値にとどまっている．

後付け条件式は実際には上の例よりももっと複雑なことがある．その式は MUMPS で許されるどんな式でもよいのであって，後付け条件は3つの MUMPS 命令（IF, ELSE および FOR で，これらは後の章で検討されるはずである）を除きすべての MUMPS 命令に追加することができる．WRITE 命令と組合わせた場合に後付け条件は状況に応じて出力に柔軟性を与えるものとなる．一例として，もし多くのデータの組みを処理する場合に，その途中の経過をモニターしながら，データ処理の各項目を書き出し，どこで止めるかをあらかじめ設定しておくことができるので思いのほか有益であることを発見するであろう．後付け条件のない場合 CRT 端末またはプリンターに無条件に多くの

量の出力があらわれてしまうことがあるからである．よい方法は処理結果のメッセージを出しながらそれを100回で止めようとすれば後付け条件次第で，それを100回で止め，それ以上を出力させなくすることができる．たとえば，変数Nに加算をおこなって出力させる場合，次のような命令をNのあとに挿入しておいても，望む出力を100回くりかえすだけですむ．

```
WRITE:N#100=0  !,N,"コウモク ハ ショリサレタ"
```

ここで後付け条件は，命令の引数のすべてに，あてはまることであることに注意されたい．後付け条件法は上記以外の多くの状況で用いられるものであるが，それらについては適切な命令が紹介されるまで，詳細についてはあと回しにしておく．

命令語と関数の省略型

MUMPS は命令語と関数のそれぞれを記述するのに２通りの方法を許している．第一には全語を完全に記述すること，たとえば，SET, KILL, WRITE, $SELECT などである．第二の方法は命令語の第１字ないし関数のはじめの２字を書くことでよい．今までに検討対象とした命令語と関数は次のように短縮できる．

KILL	K
READ	R
SET	S
WRITE	W
$SELECT	$S
$CHAR	$C
$EXTRACT	$E
$FIND	$F
$LENGTH	$L
$PIECE	$P
$RANDOM	$R

$SELECT　　　$S

省略型の命令語の例としては：

R "ドウゾ キョウ ノ ヒヅケ ヲ イレテクダサイ　", DAT

W　!!, "キョウ ノ ヒヅケ ハ", ?20, DAT

S A=3, B=A#2, C=A<1, R="XYZ"

これらの省略型では1個の文字のみみると，あたかも1文字の変数のようにみえる．はじめの行のRは命令語というより，実際変数名のようにみえる．MUMPS ではこれらの区別を省略型のあとにスペースを1つ置くことで，変数名と区別することにしている．この入門書では命令文の完全スペルを用いて混乱を避けるつもりである．

MUMPS のもう一つの特性として上述の省略型の一部ではあるが，スペースはたまたまそれが命令語にともなって存在するかしないかで重要となる．この入門書の例をみれば，たとえば，SET 命令の引数並記ではスペースを許していないが，命令語の直後の引数との間には1個のスペースがおかれていることがわかるであろう．後に述べるが，ある場合には命令文の構文を確保するために2個のスペースの必要なこともある．他の言語を知っているプログラマは，何個のスペースを挿入してもよいことに慣らされているとすれば以上の点には特に注意すべきである．

要約

算術，関数，および文字列演算子について以前の章で解説したあと，この章では命令語についていくつかの命令語と関数とを用いて一般原則を紹介した．次の章では命令行がどうやって，以上に述べた指定の組合わせでつくられるかをみることになる．

第 7 章

命　令　行

命令行の書式

今までに MUMPS 言語での“名詞”や“動詞”でどのように文を構成するかをいくつかの“言語”を紹介しながらみてきた．ここでわれわれは利用できる機能をもつような宣言文（命令文）をどのようなグループを組合わせてつくるかに目を向けてみる．

実行可能な MUMPS プログラムの基本単位を命令行 (command line) とよぶ．この単位は以前の章で紹介した要素——命令語，それと構文上結びついた変数（または変数群），演算子，そして関数などでできている．さらに，命令行は他の要素（ラベルや注釈文）を処理の制御や内容の解説のために含んでもよい．１つの命令行は少なくとも１個の命令語をもっているはずで，これまでの章で例示したものは命令行として有効なものである．しかし，１つの命令行は多くの命令語を含むこともでき，１つの命令行は端末の行の物理的な長さに制約されるのではなくて，MUMPS のもつ制限，すなわち，文字例のもつ 255 字という長さの制限に支配される（これは命令行は文字列として格納されるためであり，標準 MUMPS の移送性基準となっている）．

命令行は３種の異なった構成因子をもつ．これらは，１つの行ラベル (line label)，１つまたはそれ以上の命令語，および１つの注釈文 (comment) である．しかしこれらの因子のうち１つまたはすべてがある命令行の中では欠けていることもある．それは命令語を１つも含まない行が制御上ないし仕様上の都合で，さらにはコードの読みやすさのためにさえ設けられることがあるからである．

行ラベルは行 (line) のはじめに参照用の目的でつけておく名前である．行

ラベル名 (line label name) は変数名をつけるときの規則に従ってつくられる（すなわち，%ではじまるか大文字のアルファベット文字ではじまりアルファ数字が続く）か，または整数であってもよい．ラベルに真数を用いることは許されない．ラベルは，それ自体固有な命令行の識別のためのユニークなものでなければならないので MUMPS コードの１つの文節ブロックの中では同じ名前をもつ2つのラベルは許されない．ここで述べたこのような１つの文節については第8章で述べる．文節の中の１個の命令行にただ一つだけラベルがついていてもよく，しかもこの１つのラベルも不可欠のものではない．結局ラベルはもし付けるとすれば行のはじまりになければならないということになる．

命令行でただ一つ要求される部分があるとすれば，それは行開始文字 (ls) であり，ある装備ではスペースであり，他の装備ではタブ (tab) であったりするが，この行開始文字がラベル域と，以後の命令行との間を分離する．命令語や，あるときは注釈文が存在するときにはこの行開始文字のあとに出てくる．

注釈文（コメント）はセミコロン（；）にはじまり，そのあとプログラマがこの命令行（ときにはコードの文節）の目的を読者に申し送り理解を助けるために書き置きたいと思う自由文がある．このような MUMPS の書式ではプログラム内部に仕様を明瞭にしておき，行や文節と他の命令行や文節との関係を詳細にすることができる．MUMPS のインタープリタがセミコロンに出会うと，自動的にそれ以後の行を無視することになる．それゆえに注釈文を付けるのであれば，いつも命令行の終りに付けてその行に関係した注釈を加えるべきである．ときにはプログラマは，全部注釈文からできた行を書くこともあり，そのときは行のはじまりはセミコロンとなっている．

命令行の概念を示すために，ここで2，3の例を考えてみよう．まず次の行をみてみよう．

```
START READ "タイジュウ (KG) ヲ イレナサイ ",K SET
        L=K * 2.2 WRITE L,"ポンド デスネ？"
```

以上の命令行はラベルと3個の異なった命令語をもっている．最初の命令語はユーザーに体重をキログラムで入れることを要求し，次にこれをポンドに換

算し，終りにその計算でよいかどうかを確かめるためのメッセージを出している．ここでラベルのあと1個のスペースをおいて命令語のはじめのものが書かれ，おのおのの命令語は1つのスペースで前後分離していることに注意するのがよい．ここにはコンマを用いると MUMPS では他の意味となってしまうので，おこなってはならない．以上に出された命令行は完全なミニアチュア プログラムであり，入力と計算と出力を備えている．“行”とは物理的な1行をいうのではなく，1定数以上の文字を越えない限り1つの“行”を形成することができることに注意．

命令行でもう1つの大切なことは，それが左から右に向って実行されることである．上にあげた例で，左から右への命令語実行を踏まないならば何ら意味のないものとなり，左から右への実行順序であることは一目瞭然に理解されるはずである．後付け条件の式がある場合には（第6章参照）この式の評価が真となるまでこの命令語の実行が保留される．真でない限りこの命令文のあとの部分は無視されることになる．それでも左から右への実行順序は，はっきり保たれている．次の例では，A=0，B=7 と仮定してみよう．

SET：A　B=B＊A　WRITE　“B ノ アタイ ハ　”,B

この例では A=0 であるので SET　B=B＊A が無視される．しかし後付け条件の制限は残りの命令行にまで及ばず，MUMPS は B ノ アタイ ハ 7 と書き出す．もしAが0以外の値であれば後付け条件は真（すなわち0でないこと）となり，Bはそれに応じて変化する．

IF 命令と $TEST 特殊変数

今日のコンピュータでもっとも強力となった能力の一つは，条件次第で種々の枝分かれができる点であり，特定の条件下に特定のうごきをとるようになっている．MUMPS にはこの性質を十分に利用するための数種の方法がある．われわれはすでに $SELECT と後付け条件法を知ったが，これらは条件付きの実行の応用例なのである．MUMPS で，IF 命令の及ぶ範囲は，IF の右側にくる命令行である．たとえば，X=5，Y=3 として次のような命令行があっ

たとする.

```
IF X<Y SET Z=Y#X WRITE "Z=", Z
```

この命令行を実行するにあたり MUMPS はまず IF 命令の引数が真かどうかをみる. この例では偽であるから SET 命令も WRITE 命令も無視される. 反対に, 次のような命令行があったならば, 同じ変数値を用いた場合,

```
IF X>Y SET X=Y WRITE "X ヲ ツギノ アタイ ニ カエル", X
```

IF 命令はこの場合真と評価され, SET と WRITE の命令行は実行される. 重要なのは IF の評価が影響する範囲は同じ命令行内であることである. しかしこのときに1つだけ変化しているものに $TEST 特殊変数がある. IF 命令の引数を評価したときに, $TEST は引数である真偽値式 (truth value expression) の結果が偽 (0) か真 (1) かによって, 0か1に設定される.

$TEST は IF 引数結果の及ぼす範囲を延長拡大するのに役立つ. ここで, ある1個の IF 引数の真偽評価つまり真偽演算 (truth operation) の結果が多くの作業に及ぶ場合を仮定してみよう. これらの作業のすべてを一行の命令行内に書けない場合には $TEST 特殊変数を用いない限り, 続いて書く命令行に IF とその引数をくりかえし書かねばならない. たとえば, AGE=19, RES="カリフォルニア" と設定されている場合に,

```
IF (AGE>18) & (AGE<21) & (RES="カリフォルニア") WRITE !
IF  WRITE "アナタ ノ ネンレイ ハ", AGE
IF  WRITE " ソシテ アナタ ノ ジュウショ ハ ", RES
IF  WRITE " デスカラ, トウヒョウケン ハ アリマス ガ ビール ハ
         カエマセン.", !!
```

最後の行の終りの2文字は書式制御文字であることはわかるであろう. この例では第1行目で IF の引数が評価され, 真であることが決定し, $TEST が1となり, その行の残りが実行される. 第2行目に入って IF 命令が2個のスペースを従えて出ている. 第2のスペースは引数をもたない IF 命令であることを指定している ("引数なしの IF"). この状況では MUMPS は $TEST の値をしらべ (ここでは前の行で真値が入っている), その結果, $TEST が1

であるので，その行のあとの命令行を実行する．同様にして以下の２つの行も実行されることになる．

ELSE 命令

ある場合には IF 命令の評価結果いかんによって，１つの作業あるいは別の作業をさせることが望まれる．このときには引数なしの IF と正反対の命令語として引数なしの ELSE を用いる．つまり，ELSE は $TEST の値をしらべ，それが０である場合に，その右に続く命令行を実行させる．前掲の例の引数なしの ELSE を応用してみると：

```
IF    AGE'>18  WRITE "アナタ ハ トウヒョウ スルニモ ビール ヲ
                    カウニモ ワカスギマス." QUIT
IF  (AGE>18) & (AGE<21) & (RES="カリフォルニア")  WRITE !
IF    WRITE "アナタ ノ ネンレイ ハ ", AGE
IF    WRITE " ソシテ アナタ ノ ジュウショ ハ ", RES
IF    WRITE " デスカラ, トウヒョウケン ハ アリマス ガ ビール ハ
                    カエマセン.", !!
ELSE    WRITE "アナタ ハ トウヒョウケン ガ アリ ビール モ ホシ
                    イダケ カエマス."
```

この例は一般に 19 才以上に投票権があり，カリフォルニアだけが 21 歳までアルコール飲料の購入を禁じている住民区であると仮定している．この例では２行目の IF の引数が真であるときのみはじめの３行を実行し，そうでない場合第６行目のみを実行する．

ELSE 命令には引数をとる構文はない．IF の引数の中に，その逆の場合を実行させるための評価を含んでいるからである．実際に“引数なしの ELSE”とよぶことはなく，常に２個のスペースをともなう．

要　約

この章では命令行の概念を紹介し，その主要構成と，１行内で実行されるこ

とのできる複数の命令文について例示した． MUMPS 言語の条件付き実行のために用いる IF と ELSE も例示してこれらの命令語の及ぶ範囲が１つの行内にとどまることを解説した．

第 8 章

ルーチンの構造

第7章で命令行の性格を学んだので，MUMPS コードの構成の次のレベルに目を向けルーチン（routine）について解説する．ルーチンとは命令行の集合でそれに1個の名前をつけたものである．この名前をルーチン名（routine name)とよぶが，ルーチン名は参照に用いる変数名をつくるときと同じ規則でつくり，その名前でルーチンを格納したり，検索したり，または他のルーチンからそれを呼び出して用いる．他のルーチンからのルーチン呼び出しのための命令語は第9章で解説する．一般的な慣習として，MUMPS ルーチンはルーチン名をラベルとしてもつ行ではじまり，そのあとに注釈文として，著者，作成日付および最後に修正した日付が書かれる．

ルーチン全体の構造は一連の命令行からなり，命令行はラベルがついていたり，いなかったりする．典型的なルーチンは数個のラベルのついた命令行がところどころにあり，ラベルなしの命令行と混合した形をしている．

ルーチンの最後は（MUMPS システムの慣習として多くに用いられているが）2個の ASCII 文字で刻印してある．それらは復帰（CR）と改ページ（FF）である……ASCII 文字については付録参照……．これらの2文字に出会ったときに MUMPS はルーチンの終りに到達したと判断する．

命令行の集合であるルーチン体は，通常書かれた順序に従って実行されるが，今までにわれわれは命令語をとばし（後付け条件）たり，命令行をとばしたり（IF 命令と ELSE 命令）すること以外には，実行順序について触れていない．実際には命令行が格納されている順序通りの実行でなく種々の実行順序をとりうるような書き方が必要となってくる．こういう場合に行ラベルは新しく生じた実行順序によるはじめの行をさがすときの“ゆき先識別子（行参照，line reference)”として役立つ．

ルーチン内で実行順序を制御し，ゆき先を指定する特異的な命令語はDOとGOTO であり，これらについて以下の節で解説する．

GOTO 命令

プログラムに課せられた問題によっては，ルーチン内の多くの指示群をとびこえて他所の指示を実行させるための分岐構造を必要とすることが多い．このような，分岐点で実行のゆき先をかえることは，前章でみてきた IF 命令などによっておこなうことも一つの方法ではあるが，もっと多くの命令行をとびこえる実行の分岐のためには GOTO 命令を用いるのが，ずっと便利な方法である．GOTO ラベル名という指定で MUMPS インタープリタは，次のステップとしてこのラベル名のある命令行を実行させよと解釈する．このような分岐ロジックの例として次の例を示せばよかろう．

```
      READ !, "アナタ ノ セイベツ ヲ イレテ クダサイ (M/F)",
              SEX IF SEX' [ "F" GOTO MALE
      READ !, "イママデ ニ セイリ ガ トマッタ コト ガ アリマス
              カ (Y/N)", ANS1
      READ !, "セイリ ハ ジュンチョウ デスカ (Y/N)", ANS2
              GOTO CONT 1
MALE  READ !, "カツレイ ヲ ウケテ イマスカ (Y/N)", ANS3
      READ !, "ゼンリツセン ノ ショウガイ ニ カカッタ コトガ ア
              リマスカ (Y/N)", ANS4
CONT1 READ !, "チカゴロ タバコ ハ スイマスカ (Y/N)",ANS5
      IF ANS5[ "Y" READ "トキドキ デスカ, マイニチ デスカ,
              タクサン デスカ (ト/マ/タ)", ANS6
```

以上でみたような分岐以上に複雑な分岐する質問文を，このような GOTO ラベルの形で書くことができることは，すぐに理解できよう．実際 MUMPS の広く用いられる理由の一つは問診，質問型式，YES/NO によるアンケートなどにある点にも留意しておく必要がある．患者の既往歴問診や患者教育には

欠かせないものとなってきている．

DO 命令と QUIT 命令

DO ラベルの名形式は実行制御をゆき先のライン参照にもってゆく．しかし DO 命令は最後に制御が再び DO 命令に続く次の命令にもどすことを知っておかねばならない．DO によって制御の行き先を選ばれた命令行群の実行が終って再び実行制御を DO で中断された位置にもどすには2つの方法があり，第1の方法はルーチン体の中で残っている命令行がすべて終った（CRFF に達した）ときであり，第2の方法は DO ラベルで，とんだ実行の中で QUIT 命令に出会ったときである．第1の方法の例は次のミニアチュア ルーチンで示すことができる：

```
PCTACTIV  ; RFW 7/78, UPDATED 9/78
          ;イチニチ ノ カツドウ ニ ツイヤサレル ジカン ヲ イレ,
           % = カンサンスル
          WRITE "アナタ ノ イチニチ ノ セイカツ ノ ジョウホウ
           ガ イタダキタイデス."
          WRITE !, "ツギ ノ カツドウ ニ アテテイル ジカンスウ
           ヲ イレテクダサイ."
          WRITE !, "スイミン, ドクショ/ケンキュウ, タンレン, コ
           ウツウ/リョコウ, ショクジ/レジャー ノ 5コウモクデス."
START     READ !, "スイミン ハ ",V DO PCT SET SLPCT=Y
          READ !, "ドクショ/ケンキュウ ハ ", V DO PCT
           SET RDPCT=Y
          READ !, "タンレン ハ ",V DO PCT SET EXPCT=Y
          READ !, コウツウ/リョコウ ハ",V DO PCT SET
           COMPCT=Y
          READ !, ショクジ/レジャー ハ",V DO PCT SET
           REXPCT=Y
```

```
         SET TOTAL=SLPCT+RDPCT+EXPCT+COMPCT
           +RLXPCT
CHECK    IF TOTAL>.9 GOTO ENOUGH
         ELSE READ !,"モット ホカ ニ ナニ ヲ シマスカ?",
           OTHER
         SET TOTAL=TOTAL+OTHER GOTO CHECK
ENOUGH   IF TOTAL=>1.1 WRITE "ジカン ガ オオスギマス.
           ジカンスウ ヲ イレナオシテ クダサイ" GOTO START
         WRITE !,"アリガトウ" GOTO FINISH
PCT      SET Y=V/24
FINISH   ;ルーチン オワリ (CRFF)
```

この例は意図的にまずくつくられ，実用的なものでないが，DO ラベルの命令文でコードの分節が何度も用いられる様子を示している．ここでDO命令でPCT から最終行にゆき，そこでルーチン終了 CR, FF の続く2文字（端末にはあらわれない）に出会うと，ルーチンのはじめにもどることに注意．FINISH と書かれたラベルの行は命令語をもたないのでここでは実行される何もない．

以上のタスクを完全にする一つの方法は PCT というラベルのついた行を次のように書くことである：

```
PCT  SET  Y=V/24  QUIT
```

MUMPS インタープリタが QUIT 命令に出合うと，制御はそこから，すぐ前に用いられた DO 命令のあとに帰る．このような QUIT を用いると，ルーチンの終りの方にこうして呼び出され，再び呼び出したところにもどるまで実行するコードすなわちサブルーチンを多く書いておくことができる．

引数の後付け条件(post-conditionalized arguments)

第6章で知ったことに，命令語には後付け条件をつけて真偽値を判断してから命令の実行をおこなう構文があった．ある種の命令文（GOTO, DO, および EXECUTE）では，またそれらの引数の後に条件を付けることができる．

命令語の後付け条件の場合には条件が偽と評価されると引数部分を含み，その命令文は実行されないのに対して，引数の後付け条件では，その条件が偽と評価されてもその引数に対する命令語が実行されないだけである．その引数のあとに並記される引数はふつうどおりに実行される．次に示す2行のコードでは，入力されるデータを検証して，適当なラベルをもつ行へと制御を移すものである．

```
CHECK  READ  !, "データ ハ OK (Y/N) ?", X
             GOTO  DONE:X="Y", ENTER:X="N"  WRITE "?"
               GOTO  CHECK
```

ここでユーザーの反応が正しくないと，プログラムは疑問符（?）を出して再び次の行ではじめの質問をすることに注意．

HALT 命令と HANG 命令

しばしば自動車や機械やまたコンピュータ言語の多くの使用説明書の中で相当あとになってその停止方法を説明しているが，この入門書もこれらの例にもれていない．ここで停止という基本的な利用法を数えておかなければならない．

HALT は，引数をもたない命令語で，まさに「停止」を命ずる．これは，すべての実行を停止させ，制御をコンピュータのオペレーティング システム (OS)[訳注] にもどす．

HALT は引数をとることはできないし，また HALT に続く同じ命令行の命令語も実行されることはない（例外として HALT そのものに後付けの付いている場合は別である）．プログラマにとっては，目前の状況から一時逃避したい（コンピュータの電源をきることでなく，MUMPS を一時中止したい）ようなときには非常に気楽に用いることができよう．

訳注）コンピュータを能率よく働かせる自動制御機構で，単位時間に多重プログラムを処理し，おのおのの応答時間を短縮し，言語翻訳のモジュールを操作し，入出力装置の稼動，記憶領域の使用効率，ファイル空間の管理，入出力バッファリング，優先処理などシステム制御と処理のコンピュータに備わったユーザー アプリケーション以外のプログラムをいう．

HANG 命令は言語的には HALT 命令よりも幾分か激烈なひびきをもつが，本当はかえって人目につかない命令なのである．HANG 命令はユーザーに，指定されたある秒数間だけ操作を延期し，待ってあげる時間を許す命令である．MUMPS インタープリタは，だまって指定された時間を測定しており，その後次にくる命令文に移るのである．ほとんどの MUMPS 命令語のように HANG 命令は後付け条件法で修飾してもよい．このことを理解するために，ここでわれわれの書いたルーチンのユーザーはプリンター端末を用いるか，ときには CRT 端末を用いることもあるとしよう．その場合 CRT 端末にいるユーザーのためには，多数の行数の情報を書き出してやるとき，画面が一杯になった瞬間スクリーンのページをパッとかえるのではなく，しばらくそのページを静止させて読ませてやる時間が必要なことは知っている．こういうときには，たとえば，DEV という変数をつくっておき，ルーチンの実行の前に次のような命令行を置いておけばよいだろう．

READ !, "プリント タンマツ ヲ ツカイマス カ (Y/N)？", DEV

このようにすれば，何度も繰返しプリントをさせるルーチンにきたとき，次のような命令行を入れておくことができよう．

HANG : DEV ["N" 10……このあとの命令文に続く

この HANG 命令はユーザーが Y(ES) を入れたか N(O) を入れたかを判断し「プリント端末を使用していない」ことが判断されたときに，次の命令文に移る前 10 秒間休止させる．ここで DEV=N とせずに含有演算子（[）を用いていることに触れておきたいが，ユーザーは多くの場合 YES とか NO と書き，READ 命令中で書き出されるような指定はほとんど読まないことさえあるので，N または NO によって正確に指定された場合に，その "N" 文字によって判断できるようにしてある．

要　約

これまでの章でわれわれは MUMPS ルーチンを書くための基本的な要素をすべて学んだことになる．ルーチン内での能力を強化するために，今後さらに

2，3の命令語を追加して学ぶことになろう．この第8章で学んだことは，ルーチンが特定の識別名ではじまること，ルーチンはラベルをつけたり，つけないままの一連の命令行より成ること，またルーチンは復帰，改ページの2文字（ほとんどのシステムでは CR と FF）で終ることであった．このほかにルーチン内で制御先をかえて分岐させることを GOTO と DO の例で学んだ．

次の第9章では，ルーチン外からのルーチンへの到達法について述べる．それによって MUMPS プログラムが実行されるための要素群についてはすべて知ることになる．それについで MUMPS によるファイル管理の学習をすることになろう．

第 9 章

プログラムの構造

ルーチンは MUMPS インタープリタが1つの単位として取り出し，用いることのできる命令行の集合である．ルーチンには名前があり，慣例として第1行のラベルがルーチン名であり，そこに作成日とか作成者名ないし作成名の頭文字をつける．

コンピュータによる情報処理系をつくるときの最終的な段階は，プログラム(program)として動かすことのできるようにルーチンをうまく連結することである．DO/QUIT コンビネーションを用いて一連のルーチンを包含し，そのおのおのが特定範囲の処理をつかさどるようにした プログラム モジュールをつくることもできる．ルーチンの間で制御を移すために GOTO を用いることもでき，この場合はルーチン間のプログラムの流れのために DO/QUIT に比してそれほど厳格な制御の設計法を必要としない．

ルーチンを対象とする GOTO 命令と DO/QUIT 命令

すでに記憶域に格納されているルーチンに近づく方法は2通りある．前の章で解説した GOTO と DO がそれであるが，このときルーチン名に上向き矢印“^”(ASCII 文字表で10進コード 94)をつけて用いる．(^文字はある種の端末では上向き矢印……up-arrow……と刻んであることがある)．GOTO ^ルーチン名の形で起こる作動はルーチン内でラベル名に対して GOTO 命令をつけたときと同じで，制御のゆき先はその指定ルーチンの頭であり，そのルーチン内の命令行はルーチン実行の場合と同じである．かりにたとえば多くの請求書を処理するとして，そのためには請求源による仕分け，そのおのおのについての項目分類，金額，支払い額，最後に請求書全体からの集計表までがなされねばならないが，このうちのいくつかの作業は論理的に1つの処理個所で

おこない，他のいくつかの作業は別におこなうことになる．実際，集計表などは月末の〆切り時点でおこなうものである．こういう一連の処理系でも通常タスクをより小さいサブ ユニットに分画して問題処理にあたるのが普通である．このような小さいユニットが MUMPS でいうルーチンになる．このようなルーチン群の仮想的な構成を一部書いてみると次のようになるだろう：

```
  ORIGIN ; J. M.  9/1/76
         ; セイキュウ サキ，ヒンモク，スウリョウ シワケ
         .....
         ..... GOTO  ^DEBIT
         .....
   DEBIT ; J. M.  9/15/76
         ; セイキュウショ NR.，カシカタ キンガク ケンテイ
         .....
         ..... GOTO ^PAYMEMT
         .....
 PAYMENT ; J. M.  9/6/76
         ; シハライ プリンター シュツリョク；アテサキ ラベル ノ
           タイプ
         .....
         ..... IF  END  GOTO  ^SUMMARY
         GOTO  ^ORIGIN
 SUMMARY ; J. M.  9/20/76
         .....
```

この順序は “END” という条件が真であると判定されるまで何度もくりかえされ，“END” 条件が真となったとき SUMMARY というルーチンの実行がはじまる．

以前にも述べたように，この例はプログラムの実技上は好ましい例ではない（このような GOTO の用い方はストラクチュアド プログラミングの慣習に

反している)．しかし，一方では複雑な課題を問題別に分割してプログラムが書かれることを示している．

ルーチン間の制御のやりとりでもっと受け入れやすい処理法はDO/QUITシーケンスを用いることである．この方法ではDO命令にゆき先きのルーチン名に“^”を付け，DOの実行によって制御はそのルーチンの頭に移ることになる．そのルーチンが一度ルーチンの終りまで実行されるかまたはルーチン内の命令文としてのQUITに出会うと，このルーチンを呼び出した前のルーチンのDO命令の実行のすぐあとにもどる．DOとQUITの組合わせは作業の単位を追跡し保守するのに効果的な制御の方法といえる．上の例に掲げたのと同じルーチンで次のようなコードの順番におきかえたらどうなるだろう．

```
MASTER    ; J. M.  9/1/76
          ; セイキュウショ ショリヨウ マスター
BEGIN     ...
          IF  END  DO  ^SUMMARY
          DO  ^ORIGIN
          DO  ^DEBIT
          DO  ^PAYMENT
          GOTO  BEGIN
          ....
ORIGIN    ; J. M.  9/15/76
          .... QUIT
DEBIT     ; J. M.  9/10/76
          .... QUIT
SUMMARY; J. M.  9/20/76
          .... QUIT
```

以上の順序をみると主プログラム（マスター）の中に全プログラムの流れが書いてある．各ルーチンの中にある詳細に触れなくても，基本的な主要ステップはこのマスターの中で変化させることができる．ルーチンには1つ以上の

QUIT をもっていてもよく，プログラマが分岐を選択してルーチン内での一部分を実行させ，その部分が終了したら元のところにもどすことができるようになっている．

MUMPS ではある1つのプログラムのすべてのルーチンがはじめからコンピュータ内記憶装置に入っていることを必要としていない．別のいい方をすれば，MUMPS はルーチンのおのおのについて必要なときどきに，ユーザーの作業用パーティションにもちこんで（load して）くれる．このために必要なつどディスクから呼び出す装置や，ある装置はバッファー領域をもっていて頻回なディスク アクセスを避けられるようになっているものもある．いずれにしてもプログラマにもたらされる最終的な効果は，彼が大きく複雑なプログラムをつくる意図をもつにせよ，もたないにせよ，おのおののルーチンを1回のもちこみ (loading) のための1つの単位として十分に小さくまとめるか否かにかかっている．このような MUMPS の特性はモジュール性を高める方向に役立ち，このことが複雑なプログラムを補修し保守するために大きく役立ってくれるもととなる．

ラベル名とルーチン名は，ともに GOTO と DO の引数として用いることができる．GOTO ENTER^ABC では ABC というルーチンを呼び出し，そのはじめからでなく途中の ENTER というラベルのある行から開始させる．借方 (DEBIT) と支払い (PAYMENT) が上の例で DEBIT というルーチンに入っていたように同じルーチン内にある方がよい場合には，マスターで次のように書いた方がよいと思われる．

```
MASTER  ; J. M.   9/1/76
        ; セイキュウショ マスター ...
 BEGIN  IF  END  DO  SUMMARY  QUIT
        DO ^ORIGIN, DEBIT^ACCNT, PAYMENT
           ^ACCNT  GOTO  BEGIN
```

ここでは DEBIT と PAYMENT は ACCNT というルーチンの中のラベルになっていて，DO 命令の引数並記法で一行の命令行をつくった．

パッケージ(package)

以上で MUMPS の実行用命令文をみてきたが，命令語にはじまり命令行，そしてルーチン，プログラムという順でより大きい処理系まで進んできた．このような統括の仕方でもっとも大きい変化に富んだデータの集合に対する操作を可能にする大きなプログラムのまとまりをつくることができる．このように大きくなったプログラム単位のよび方として MUMPS の慣習ではパッケージということになっている．誰でも想像できようが，パッケージとは一連のルーチンから成るプログラムが集合したものと考えてよい．プログラムとは通常1つの処理単位としてのルーチン群を入れたものであり，これらを集合してより広いプログラム群によって管理体系などの技術的な目的達成の処理系を完備したものをつくったのをパッケージとよぶ．たとえば病院の検査室は，生化学，微生物学，血液学といったセクションに分かれているが，これら個々のセクションはプログラムで運行されうる．検査室全体の運用は上記のこれらの全セクションを含み，さらに検体の収集法，結果報告書作成法，会計処理，在庫管理法，患者識別法など一貫した情報の流れをもつことになる．これらすべての整然とした情報の流れをトータル システムとして提供する MUMPS ベンダーは，これらのプログラムの統括をパッケージという名でよび，それにシステム名をつけている．以上のように一つの特異的で全体包括的な情報処理系として“MUMPS パッケージ”は存在し，これらがテレビ局やラジオ放送局のスケジュールを自動化し，企業の経理大系を支え，多くの幅広い原理で活動する組織での応用パッケージの母体となっている．

要　約

この章を終えたときに，われわれは MUMPS 言語を用いて単純な問題を処理するプログラムを書くために必要なすべての要素を学習したことになる．残りの章では読者に MUMPS 環境でのデータの種々の分類法と格納法，そして検索法を紹介し，これまでの応用プログラムの活用力を高める手段と概念へと

入門させてゆくこととなる．MUMPS のもっとも ユニーク な性格はそのファイル マネジメント法であり，この MUMPS の能力こそがこの言語を効果的に活用するために特に重要な鍵となる．次の第 10 章から第 12 章までを読者が学習完了したとき，言語のプログラミング面についてはすでにすべて入門し終えることになるが，さらに本入門書では，それ以後に入門書として必要性をこえた言語上の拡張部分に触れることとし，意識的に本入門書の最後の部分にそれを置くこととした．

第 10 章

ローカル配列

はじめに

前章までに扱ってきた変数は単純変数 (simple variables) とよび慣わされているもので，変数名があるとその値が１つあるという１対１の対応をもつ変数と値との関係に立っていた．多くの場合に単純変数は便利なものであるが，ときによってはデータを幅広く集合的に扱いたいこともある．例として職員の名前などがそれである．職員の固有の名前に合わせて単純変数をそれぞれに与えることはもちろん可能であるが，このようなことをするときにはプログラムは極端に奇妙な処理方法を考え出さねばならない．MUMPS は，他の多くの言語と同様にこの問題を，言語の一部として配列 (array) を用意することで対処している．

１個の配列とは１個の単純な名前をもつが，それが複数の値をとりうるようにした変数の形式である．この集合から特定の１個の値を参照するには変数名に添字 (subscript) を付けてそれを可能とする．コンピュータ端末装置の大多数のものでは半行下げた添字 (たとえば X_2 といった形) を書くことができないので，MUMPS では他のコンピュータ言語と同様添字を括弧内に入れて変数名のすぐあとに付け，A(1) という形式で配列を表現する．この添字付き変数 (subscripted variables) を用いると，われわれは職員名簿の問題を容易に単純化できる．１人目の職員を A(1) に，２人目を A(2) にそして３番目の職員を A(3) に割り付けるといったように全部の職員名簿を格納することができる．さらにこの概念を拡張して添字に定数をあてる以外に，MUMPS は添字の場所に式を与えることを許している．これによって，次のようなルーチンの一部を用いて職員に与えるナンバーをつくらせて配列をつくることが可能となる．

```
........................
     SET  N=0 ; ショクイン ノ スウ ヲ N デ ルイケイ スル
RD   READ !,"ショクイン バンゴウ ヲ イレナサイ (オワル トキ ハ
     CR)", NO
     GOTO:NO="" DONE  ;CR ノ ニュウリョク デ 'DONE' ニ
     トブ
     SET  N=N+1, EMPNOS(N)=NO GOTO RD ; ツギ ノ ショ
     クイン バンゴウ ノ ニューリョク ニ モドス
DONE WRITE  !!,"イマ イレタ ノハ", N, "バン ノ ショクイン デ
     シタ."
........................
```

以上はルーチンのごく一部であるが，この形のプログラミングで添字付き配列を別の変数ないし式で参照させることができることがわかろう．このことはプログラマの仕事を大いに簡素化するのに役立つ．この章の後半では，もっと言語の添字概念をみることにしたい．

FOR 命令

配列にいったんそれらの値が格納されたあとは，それらの値は随意に検索し操作することができるようになる．配列の順に従って値を扱うためには MUMPS の FOR 命令を用いることになろう．FOR は命令行を反復して実行させる命令語である．もっとも一般的な形の FOR 命令に次のような構文規則をもつ．

FOR　INDEX=P 1 〔, P 2,....〕(このあとに反復的に実行される命令行がおかれる)

ここでINDEXは変数名であり，P 1，P 2,.... は FOR パラメータ（FOR parameter）である．FOR パラメータは次のように定義されよう．

FOR パラメータはインデックスとなる変数に順次値を割り付ける．新しい値が割り付けられるたびごとにそのあとの命令行が1回実行される．FOR パ

ラメータが何らかの方法でなくなるまで，または命令行の実行が中断されるまでこの実行は繰り返される（後者の例は実行中に QUIT 命令または GOTO 命令に出会うような場合である）.

このことを数種の例で示してみる．もっとも簡単な FOR パラメータは単純な式で表現される．この場合インデックスとなる変数にその式の値が割り付けられる．次の命令行が1回実行され，FOR 命令が次の FOR パラメータの値に対して同じ命令行の実行をおこなわせる．たとえば，

```
SET  S=“”
FOR  X=“A”, “B”, “C”  SET  S=S_X WRITE  !,S
WRITE  !,“オワリ”
```

とすれば，実行結果は次のようになろう．

```
A
AB
ABC
オワリ
```

上の例でXはインデックス変数であり，それに次々と “A”, “B”, “C” という値が FOR パラメータに指定されたように割り付けられ，すべての FOR パラメータが処理されたときに FOR 命令のある次の命令行が終って，制御が次の命令行に移ったことになる．

次に残りの FOR パラメータの2形について述べるが，これらは数式を必要とするものである．

numexpr 1 : numexpr 2

この場合インデックス変数（index variable）には最初 numexpr 1 の値が割り付けられる． numexpr 2 の値が次回にインデックス変数に加えられて，右側にくる命令行が再び実行される．この過程は上記の形では際限なく繰り返され，右側にくる命令行の実行が何らかの方法（下記の FOR 内の QUIT 参照）で中断されなければいつまでも，MUMPS の許す最大値までインデックス変数が達してエラーの生じるまでそれが続行されるだろう．次のような実例

で出力装置の上に偶数を書かせたら，それは終るところなく偶数を書き続けることになる．

```
FOR I=0:2 WRITE !, I
```

実行結果は，

```
0
2
4
6
....
```

これを中断するには外部から止めてやる(端末の電源を切るか，MUMPS 装備のもっている強制中断の命令語を使うかなど）以外に方法はない．この例では上記の numexpr 1：numexpr 2 の形がわずらわしいものとして示されたが，実は非常に利用価値のある目的で用いられることがあることを憶えておくのがよい．

FOR パラメータの第3形式は無限ループをつくらないという点を除いては，前記の形式によく似ている．この形式は3番目の数式を付けて，いつ右側にくる命令行の実行が終るかを指定するものである．ずっと前の職員番号の例をしらべると，N 個のデータをもっているはずの配列 EMPNOS で，NO が正しい値かどうか検定するには，次の短いルーチンを付ければよい．正しい番号ならばインデックス変数 I が EMPNO の配列にある職員番号を指示するし，正しい番号でないならば I はゼロとなってプログラムをそのまま前へもどしてくる．

```
CKNO  FOR I=1:1:N  IF  EMPNOS(I)=NO  GOTO  DONE
        SET  I=0  QUIT
DONE  QUIT
```

このプログラムでは NO に割り付けられた値が誰かの職員番号と一致するか，もしくは全職員の番号と一致しないことになるまでループを続ける．第3の numexpr 3 (N) では，I に割り付けられる数値の上限を指定している．

この例でははじめIには1が割り付けられ，最初の職員の番号が NO と比較される．これらが一致していたときに制御は DONE というラベルの行に移され，一致していないときには命令行が全部実行されてから実行が FOR パラメータにもどる．そこで FOR パラメータの第2の式がIに加えられ，次の職員番号がしらべられる．ここでIの最終値に1を加えるとそれがNという制限値をこえるようならば，その直前に制御は次の行に移り，Iを0に設定して QUIT し，NO がどの職員番号にも一致しなかったことを示す．

FOR パラメータが数式を伴ってくる2種の形式のいずれにおいても，増分パラメータの値は正の値でも負の値でもよい．増分パラメータが負の値をとる場合，無限ループをつくった前掲の例では書き方はほとんどかわらないが，3つの式によるパラメータ群の場合は第3式の意味が反対になる．第2式すなわち増分パラメータが負の場合，次々とこの値が第1式すなわちインデックス変数から引かれていって，もう一度引くと第3式（制限値）より小さくなる点に達する．

たとえば，

FOR I=9:−2:4

の場合Iが9，7，5を取る間は第3式より大であり，5から2が引かれて3となると制限値より小さいので右側に続く命令文は実行されない．インデックス変数の初期値が制限値より大である（増分パラメータが正の値のとき）場合と，制限値より小である(増分パラメータが負の値のとき）場合には，FORを含む行は実行されない．したがって，

FOR I=10:1:5 <命令行>

および，

FOR I=2:−1:4 <命令行>

では命令行は無視される．これらの場合Iが 10 または2に設定されるところまでは実行される．

FOR パラメータの種々の変形が FOR 命令の中でみられることがある．たとえば，

FOR I=0,2:1:5,8,11,−1,15:−1:13 WRITE I,“,”

では書き出される結果は，

0,2,3,4,5,8,11,−1,15,14,13,

である．

同一行で1個以上の FOR 命令を実行させることができる．FOR 命令の範囲は FOR 命令の残りの行全体を含むので，もう1つ右側にくる FOR 命令は左側の FOR の中でネスティング（入れ子，nesting）を形成する．たとえば次の命令行，

```
SET S=“”
FOR I=1:1:3 SET S=S_“A” FOR J=1:1:I SET S=S_“B”
WRITE !, S
```

では結果は1本の出力行として，

ABABBABBB

と書き出される．

ここではインデックス変数Iで制御される外側 FOR ループの範囲の中に，他のインデックス変数Jで制御される FOR ループが入れ子になっている．そこで外側ループの1回実行ごとに内側ループは完全に一巡し，外側ループの増分パラメータが1つインデックス変数に加わり，内側ループがはじめからまた一巡する．その結果上の書き出しとなる．

MUMPS ではさらに，命令行の実行によってインデックス変数を変化させることを許している．たとえば，

FOR I=I:2:10 WRITE I SET I=I−1

は FOR の増分パラメータ2がIに加えられる以前にIの値が変化して，

12345678910

という結果を書き出す．

この FOR 命令行内の機能は注意して用いないと予想もしない結果を出すことがある．

FOR 命令行における GOTO 命令と QUIT 命令

職員番号をしらべる例で示したように，FOR 命令を FOR パラメータの完了前に中断させるために，FOR 命令文内に GOTO 命令をおいたことを思い出してみよう．そこでは，GOTO によって命令行のすべての実行が中断されるので，その行にあるいかなるFOR 命令も中断するはずである．QUIT 命令は１つの FOR 命令を中断するために用いられるが，GOTO の場合とやや異なった動きをする．すなわち，QUIT が１位の FOR 命令の範囲内で実行されると QUIT を含んでいるもっとも内側の FOR 命令の実行が中断される．これを示すために上の例を少々変形して示すと，

```
SET  S=" "
FOR  I=1:1:3  SET  S=S_"A"  FOR  J=1:1  QUIT:J>I
  SET  S=S_"B"
WRITE  !, S
```

これを実行すると，

```
ABABBABB
```

となる．内側の FOR ループの無限ループを終らせているのは J が I の値をこえたときの QUIT であることに注意．ここで，この QUIT は外側のループを中断させることはなく，それはインデックス変数 I によって最後まで制御される．

配列の解説

配列の概念をさらに深く学習するにさいし，FOR 命令が配列の操作にとって強力な道具となることを知ることになるが，MUMPS 言語の配列に関してここで少し説明を加えておこう．プログラム言語のほとんどのものでは，ある配列の項１と項 10 に値を割り付けることができるためには，あらかじめ項１と項 10 との間に値の格納庫となるスペースをとっておかねばならない．すなわち項２から項９までの配列子を含んであらかじめ配列の宣言をおこない，コ

ンピュータにスペースをいくつ用意すべきかを教えておくのである．このような配列を“稠密 (dense)”配列とよび，プログラムの用いるデータのために必要となるすべての添字付き変数 (subscripted variable) がすべて用意されており，プログラムによってそれらの添字付き変数の取る値があとから割り付けられることを意味する．

MUMPS は配列のつくり方で，従来のコンピュータ言語と2つの重要な点で異なっている．第1には，配列が“稠密”である必要がなく，ただ値の割り付けられた変数のみが格納スペースをもつ“分散 (sparse)”配列を許すことである．第2にこれらの格納スペースは前もって指定し確保しておく必要がないことである．MUMPS では新しい変数に出合うごとにその格納域を見つけてその値を割り付ける．分散配列[訳注]はこのように格納スペースを節約し，用いないかもしれないスペースを予約するための宣言文をあらかじめ書く必要をなくしている．なおこれ以上に高級な技術的可能性をいうならば，分散配列では単なる数字の順序以外に，添字にもっと意味のあるものを選んで用いることもできるようになる．職員番号をしらべるプログラムの例にもどっていえば，職員番号そのものを添字として用いることもできるので，データすなわち氏名，住所または月給高などの値は職員番号を鍵として参照することもできるようになる．

多元分散配列(malti-dimentional sparse arrays)

これまでにあげた例はすべて1元の添字をもつものであった．MUMPS は多元の添字を許すものであり，実際 MUMPS の添字付き変数では添字の数を無制限につけることができる．この際添字の数は1行に入る字数によって制限をうける．たとえば，職員名と住所を入れる1つの配列を次のようにつくることができる．

EMPNA (EMPNO, 1) に氏名を入れる
EMPNA (EMPNO, 2) に町名・番地を入れる

訳注） 分散配列: sparse array はとび石配列ともいう．

EMPNA (EMPNO, 3) に県名・市名を入れる

EMPNA (EMPNO, 4) に郵便番号を入れる

この例で，EMPNA は2元配列（マトリックス）であり，第1元は職員番号をキーとしてもち，第2元は各職員の4つに分けたデータ項目をもっている．これ以上に複雑で，添字数が2つに限定されない多元の配列をつくり，職員個個の学歴，学位，就職歴など，そしてそれらの日付をもつものをつくってみることができよう．

添字に許される値

標準 MUMPS では添字として許される値には，式 (expression) として表現しうるものなら何を用いてもよいことになっているが，標準 MUMPS の互換性のための規定では，添字は“負でない整数”となっている．この制限は，MUMPS 機種のほとんどに援用されている．ときには負の値，非整数，さらには文字列を添字として許す機種もある．自分の用いるシステムについて各自その仕様書を参考にするとよい．この面での将来の標準 MUMPS への拡張路線は第 16 章に述べられている．

データ値の割り付け

データ値は MUMPS 配列のうちのどこに割り付けてもよい．値を配列の“最下位”の添字に割り付けるというような規則はない．しかしながら，ファイルを有効に利用するために，なるべく“上位”のレベルにデータの仕様に関するものがあつめられる．職員ファイルの構成を次のようにかえてみる．

変数	値
EMPNA (EMPNO, 2)	職員所属課

このようなファイルはファイルそのものが自己仕様をおこなっているので，利用者が上位 (ascendant) の値をみて下位 (descendant) に何があるかというファイルの性格を把握するのにすぐれている．

ローカル変数(local variable)

この章で解説してきた添字のない単純変数や添字をもつ変数，そして多元分散配列としてファイルを構成する変数を含めて，プログラムとともにコンピュータ内の1ユーザーの作業領域内に存在するものをローカル変数と総称する．これに対してディスク内にあって多くのユーザーが共用できるグローバル変数があり，それは第12章で解説する．

配列とともに用いる KILL 命令

KILL 命令（第6章）は配列を消去するのにも用いる．引数なしで用いるとKILL はすべての変数を添字あるなしにかかわらず消去する．選択的 KILL は特定の変数のみならず配列の特定のものを消去する．KILL EMPNOS(3) によってその値が消去されるのに対し，KILL EMPNOS では全配列が消去される．KILL の選択型式を用いて，ある特定配列を除くすべての変数を消去することもできる．それは KILL (EMPNOS) という形であるが，この場合に EMPNOS(3) のみを除いてすべての EMPNOS 配列を消去するために KILL (EMPNOS(3)) とすることはできない．配列の一要素のみを残そうとするこの試みはエラーとして扱われる．

KILL 命令にともなってもう少しつけ加えて知っておくべきこととして，どのような名前も同時に添字付き変数として，または添字なしの変数として用いられる可能性があり，またある添字付き変数には同じ変数名でも添字の数が異なっていることがあるという事実がある．たとえば，A，A(1)，A(1,2)，A(9,2)，A(27,3,4) という変数は MUMPS 環境では同時に存在しうる変数である．この場合に KILL A と書けば添字付きのAと添字なしのAはすべて消去される．これと同じように KILL A(1) は A(1) と A(1,2) を同時に消去する．すなわち参照された変数とその下位配列を全部消去する．ここで，KILL A(1,2) とした場合 A(1) の値は作用をうけないが，その下向きポインタは消される．これについては次の節の $DATA で解説する．

配列操作のための関数

MUMPS は分散配列を操作するための 2 種の関数をもっている. 値の定義されていない変数を参照することは MUMPS ではエラーとなるので, たとえば A(10) という変数の値を参照する場合にそれが実際に存在するかをみておかねばならない場合が多い. $DATA 関数は, どのような変数についても値の定義されている状態についてしらべるために用いることができる. $DATA の引数は変数名で, それは添字を含んだものであってもよい. $DATA の返す値は次の表に示してある.

$DATA の引数	返される値	意味
変数	0	定義されず, 下位配列なし
変数	1	定義されていて, 下位配列なし
変数	10	定義されず, 下位配列あり
変数	11	定義されていて, 下位配列あり

この表で, 下位配列とは引数となった変数のもつ添字と同じ添字をもち, さらに余分の添字が続く添字付き変数を意味する. 例として前節にあげた A(1, 2) は A(1) の下位配列である. 同じように, A(1,2,4,9) は A(1,2) の下位配列である. 下位配列を KILL することは上位配列にも影響を及ぼし, かりに A(1) と A(1, 2) を定義された変数Aのすべてとすれば A(1, 2) を KILL すると $DATA A(1) は 10 を返す. これは前の A(1) に下位配列がもうないからである. また, ここに A(3,7) があり, A(3) は存在しないとすると, $DATA A(3) は 10 という値を返す. すなわち A(3) は定義されておらず, しかも下位配列は存在するからである. 装備によって添字の範囲を拡張している場合でも, $DATA はこのように作動する.

$DATA を用いて添字付き変数に下位配列があるか否かを確かめることができたら, 次の論理的要求としてどうやってその下位配列の添字値を知るかという命題がある. $NEXT 関数は同一のレベル (level) の添字値を数値的な順で書き出すのに用いられる. $NEXT の引数は添字付き変数で, 最後の添字を

整数とするものでなければならない．最後の添字として −1 という信号が特別の用途のために用いられる．−1 はその添字レベルでの最小の添字を知るために用いる．$NEXT の返す値は引数の最後の添字レベルでその添字よりも次に大きい添字の値をもち，値が定義されているかまたは下位配列をもっている添字付き変数の添字であり，そのような添字がそのレベルにはもう存在しないときには −1 を返す．

この複雑そうにみえる関数を理解するためには，特定の例を用いるとよい．かりに変数Xが添字付き変数として次のものに値をもっているとする．

X(1)，X(1,2)，X(1,4)，X(1,5,6)，X(2)

ここで $NEXT を用いて次々と下記のような変数を引数にあてた場合の結果は，

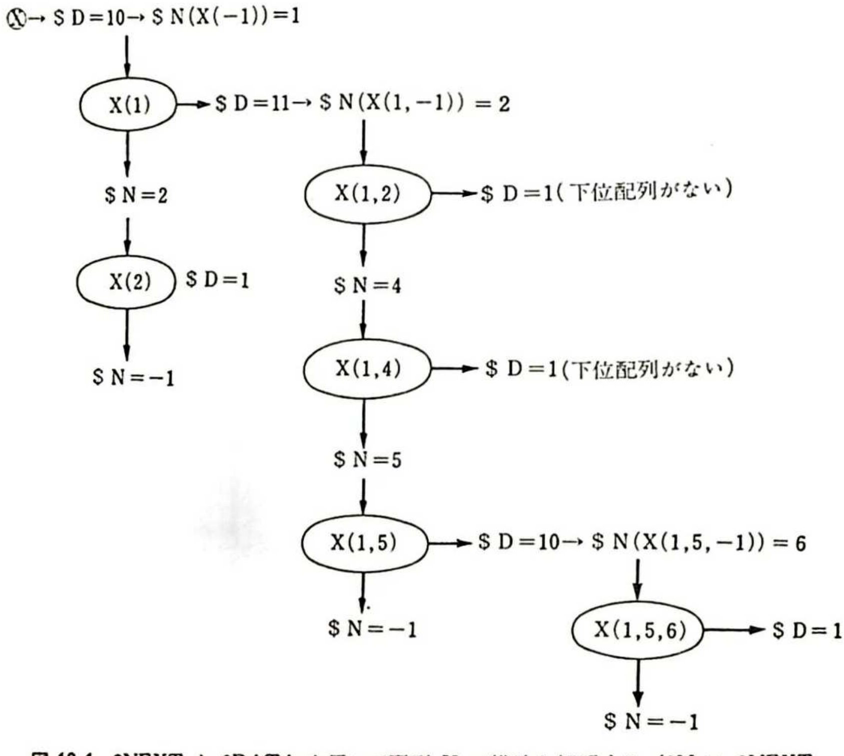


図 10·1 $NEXT と $DATA を用いて配列 X の構造を解明する（$N は $NEXT の，$D は $DATA の省略型である）

$NEXT の引数としてあてる変数	$NEXT（変数）の返す値
X(−1)	1
X(1)	2
X(2)	−1
X(1, −1)	2
X(1, 2)	4
X(1, 4)	5
X(1, 5)	−1
X(1, 5, −1)	6
X(1, 5, 6)	−1

この例にみられた $NEXT の結果と以上の説明を注意して読めば，$NEXT の種々の活用法についてほとんど知りつくすことができよう．

要　約

この章でわれわれは“分散”配列という MUMPS の特徴である添字付き変数の概念を学んだ．特定の変数があっても，それがあらかじめ宣言された稠密な配列ではないので，どんな添字をもっているか，そしてそれらに値が割り付けられているかを予知することはできないので，$DATA と $NEXT の2関数で1つの配列を完全に把握することができることも学んだ．

第 11 章

中 級 の 入 出 力

第6章では端末での READ と WRITE 命令の機能について解説した. MUMPS はユーザーが1つの端末以外に他の装置に平常の入出力 (I/O) 操作をすることを許している. 他の装置としてライン プリンタ, テープ装置類がある. ここでは同じ READ と WRITE の命令が用いられるが, 次に述べる若干の拡張をした構文規則による.

OPEN 命令

1つの I/O 操作で何かの装置ないしファイルが参照される場合には, その前にそれらの装置ないしファイルがプログラムに対して割り当てられていなければならない. MUMPS システムでは通常多数のユーザーが同時に別々のプログラムを用いているので, 装置類の利用もこれらのユーザーに整然と資源の共用がなされるようになっている. 装置の割り当ては OPEN 命令でなされる. OPEN 命令の引数はどれも次の3つの項からなるが, その第1の部分のみがここで要求されるものである.

1) 装置ないしファイルを指定するための式を書いて, それを開くことを示す装置指定式 (device specifier)
2) 装置パラメータ (device parameter) とよばれる1つまたはそれ以上の式で, 装置ないしファイルの状態を記述する.
3) 時間制限 (time out). プログラムが装置ないしファイルを割り当てられるまで待っていてもよい最大限の時間をあらわす式.

I/O 操作の性格はその装置の性格や MUMPS 装備の方法によって非常に変化がみられるので, ANSI 標準は 1) の装置指定式についても装置パラメータの式についても規定はしていない. 自分の用いている MUMPS 機種のこれら

の詳細はよく仕様書で学ぶのがよい．この章では，装置パラメータの詳細などについては言及せず OPEN 命令の利用に焦点をおいて解説する．

OPEN 命令がプログラム上にあらわれると，MUMPS はこのプログラムにその指定する装置ないしファイルの排外的な“占有権 (ownership)”を与えようとする．このことで2人のユーザーが同時に同一装置を使おうとする混乱が避けられる．OPEN 命令に出会うとユーザーがさらに指定するプログラム コードはこの占有権が確保されるまで一時棚上げされている．この間に時間制限が OPEN 命令の第3項に示されていれば，その時間が過ぎると制御はプログラムのユーザーの指示にもどされる．この時間内に占有権が確保されれば $TEST 特殊変数が真という値をもつことになり，これで OPEN が成功したことを示す．誰か他のユーザーがその装置を用いているときに，時間内に占有権が確保できないときは $T の値は偽に設定される．ユーザーは占有権が得られたかどうかをこのように $Tの値でしらべることができる．時間制限が書いてないときには$T は何の影響もうけず，占有権が確保されたときにのみ OPEN 以後のコードに制御がもどされる．

以上のことを示すために，現在われわれがユーザーの端末 (主装置 principal device) を0とするような装備を用いているものとして，ライン プリンタが1，カード リーダが2とそれぞれ装置指定がなされているものとする．自分の端末に対して OPEN 命令をかけることはほとんどの装置で必要はない．しかしプログラムでライン プリンタやカード リーダ（これらは，他のユーザーとも共用している）の必要な部分があれば，それらを OPEN することになる．その際の書き方は，

OPEN　1:dl, 2:dr

となる．ここで dl と dr とはライン プリンタとカード リーダの装置パラメータであって，ここでは時間制限は指定されていない．ここで，ライン プリンタのためにはいつまでも待つが，カード リーダのためには1分以上は待てないというつもりならば，上の命令行を修正して，

OPEN　1:dl, 2:dr:60

として，最後の 60 という値が 60 秒の時間制限をあらわす部分となる．このOPEN に出会うと MUMPS の特殊変数 $T はカードリーダが使える状態になったかどうかをテストしはじめることになる．

USE 命令

いったん装置ないしファイルが OPEN されると，次にはどの装置を実際用いるかという決断をして READ ないし WRITE 命令でそれを参照することになる．ユーザーは常に端末から MUMPS と情報交換をしているので，OPEN 命令をかけておいても，実際にはどの特定装置を用いることになるのかまだわからないこともできてくる．ここで，現用装置 (current device) という概念が必要となってくる．現用装置とはいつでも READ 命令と WRITE 命令によって参照できる単一の装置であり，通常現用装置とは端末を指す．プログラムで現用装置を変化させるときには，USE 命令を実行させる．USE 命令の各引数に参照されるべき装置を指定し，おそらくそのあとに装置パラメータをつけたものとなっているはずである．この装置パラメータも，実は言語装備者によって相違のあるものとなっている． USE 命令の各引数が実行されると，そのときこれらの装置なりファイルなりが現用装置となる． USE 命令には複数の引数をつけても，最後の引数しか次にくる作動にとって実際の効果をあらわさないから，無駄なことである (装置パラメータについてはこの限りではないが)．USE 命令に関連して $IO という特殊変数が用意されており，これで今何が現用装置として割り当てられているかをみることができる．USE 命令の実行は READ と WRITE の実行による I/O (入出力) 操作がこれからその装置に対しておこなわれることを前提としているので，このときの USE 命令の引数はすでに前に OPEN によって占有を確立した装置をもっていなければならない．

以上の例をそのまま続けてプログラム例とする場合，

USE 1

という命令は，このあとにくるすべての READ および WRITE 命令をライ

ン プリンタに向けて指令させることになる．このときには $IO は 1 に設定されているはずである．この状態は次に別の USE 命令が実行されて他の装置が指定されるまで維持される．さて，ライン プリンタに対して出される READ 命令は意味をなさないことはわかるであろうが，現用装置がライン プリンタである場合，それに対して READ 命令が出されないような注意をすべきである．

ファイルないし装置のおのおのについて，特殊変数のそれぞれ $X と $Y とが準備されている ($X, $Y については第 6 章参照)． また書式制御 (!, #, および?) はすべてのファイルないし装置において定義されているが，これらが実際にはどんな効果を出すかは装置提供者ないし装備者によって定義されるものである．

CLOSE 命令

いったん特定の装置を含む I/O 操作が完了したらユーザーはその装置の占有権を放棄しなければならない．装置を開放することは CLOSE 命令でおこなわれ，この命令は OPEN 命令や USE 命令のように CLOSE する装置のおのおのを引数としてもつ．そこに，装備者が定義する装置パラメータを付加することも可能である．現用装置占有権 (current device ownership) を放棄したときの通常な動作は，欠損代用すなわち通常の端末に帰ることである．標準 MUMPS は，必ず通常の端末に欠損代用として復帰することを要求してはいないので，ある種の装備では，占有権の放棄後に起こる動作がこれとやや異なる場合もありうる．

READ 命令と WRITE 命令の拡張

この章でカバーする最後のテーマは READ と WRITE の拡張型である．第 1 の拡張型は READ ないし WRITE が単一文字を受信したり発信したりする機能である．一文字 READ は READ の引数となる変数名の前に星印をつけることで指示される．

READ *XYZ

この命令では現用装置から受信された第1字が，装備者に定義された規則によって解釈され，この入力文字に対応した1つの整数値が返されることになる．これと反対に，

WRITE *XYZ

では XYZ に入力される整数が装備によって定められた規則で1個の文字に変換され，それが現用装置に書き出されることを命令している．この拡張命令はときとして特定の機能を出力装置におこなわせる場合や，特殊な通信用プロトコールを装備するためにも有用である．

READ 操作で装置から入力される信号を際限なく待っていることをなくするために READ 命令にも時間制限の拡張をもうけて OPEN 命令の解説でみたようにして用いることができる．時間の限度は秒で測られ，READ の引数のあとに付加する．命令行の例として，

READ !,"アナタ ノ ID ハ?",ID:10

は 10 秒以内に適当な ID 番号を入れないと，それまでに入った文字のみが MUMPS から返される．この間特殊変数 $T の値が時間制限の状態を反映している．1文字 READ では結果は装備が規定する整数であるが，そこにまだ何も文字が入っていないと読み取った値として −1 が返る．

$JUSTIFY 関数

第6章で，書式制御文字"?"で特定の文字位置を指定して出力欄をつくるときの方法を学んだ．文字列の第1字を合わせて欄をつくることを"左詰め"という．ときには文字列の右端を合わせて項目を欄内に書くことも有利なことがある．これは"右詰め"であり，これは数表や金額の表でよく用いることである．次に第 20 欄に変数Xを右詰めにして打ち出す例を考えてみよう．

```
FOR I=0:1:20-$LENGTH(X) WRITE " " ; スペース
ノ ツメモノ
WRITE X
```

この方法では大きいリストを書き出すときに毎度スペースの詰め物を左側において実際のXの値を書くために，時間がかかりすぎることになるので，MUMPS では $JUSTIFY 関数が用意されこの詰め物をおこなうようになっている．$JUSTIFY (X, L) は文字列式であるXを全文字数 L という領域の中に左側に詰め物としてスペースをおいてもっている．もしもXがLより長い文字列であればXそのものが返され，詰め物は施されないので，文字が失われることはない．$JUSTIFY を用いて上記の例を書くと，

```
WRITE  $JUSTIFY (X, 20)
```

となる．ここで 10 個のスペースをもつ文字列をつくろうとするときの便利な方法は次のようになることも知っておくとよい．

```
SET  X=$JUSTIFY ("     ", 10)
```

ここでオプションとしての $JUSTIFY の第3引数がくると，$JUSTIFY は数量上の修飾をすることになる．$ JUSTIFY (X, 10, 2) ではXを数値解釈したのち，小数点以下2桁以下四捨五入したあとそれらを左詰めにし，右にスペース詰めしてもっている．整数除算演算子（＼）では数量を小数点以下切り捨てることを学んだが，$JUSTIFY はそれを補充して四捨五入の演算をおこなっている．$JUSTIFY の第3引数は負でない整数でなければならない．第3引数が0と置かれたとき，それは1の桁が偶数となる整数として丸められる．第3引数の負の値は現在のところ規則違反となる．しかし標準 MUMPS 委員会はいつか将来この関数が負の値に対しても何らかの作用をもちうる余地を残している．以下の例で $JUSTIFY のいくつかの用法を示す．

式	結果の値
$JUSTIFY (3.716, 0, 0)	4（小数点がないことに注意）
$JUSTIFY (3.716, 0, 1)	3.7
$JUSTIFY (3.716, 4, 2)	3.72
$JUSTIFY (3.716, 0, 5)	3.71600
$JUSTIFY (7, 0, 3)	7.000
$JUSTIFY ("ABC", 0, 0)	0

要　約

本章では READ 命令と WRITE 命令のいくらかの拡張型によってユーザーが自らの端末以外の装置を用いうるような方法を紹介した．これらの拡張型は高速ライン プリンタなどで書き出される報告書のプログラムを書くときなどに必要となる．またこれらは MUMPS システムに大型ファイルや大量のデータを読み込むときに必要となる．OPEN, USE, CLOSE の命令ではこれらの作動を効果的にするために，装備に特有な仕様を用いる必要があるので，これらの命令を用いるに先立って仕様をよく知っておくことが大切である．

第 12 章

グローバル変数

第2章で変数という概念が紹介されたが，それは値をとる1つの独立した名前であった．この概念は第 10 章で配列という概念にまで展開された．この配列は関連するデータをグループ化して，樹枝に房がつくような集合にまとめるのに便利である．ここでは，ユーザーがコンピュータから去ってしまってからも MUMPS の中に保存される変数の原則について解説する．

現在までに本書で用いた変数は“ローカル変数”という類いの記憶域に格納されていた．MUMPS でローカルという場合一種独特の意味があり，それはアクセスする場合にも，その持続時間にも限界があるということである．MUMPS のローカル変数はただ1人のユーザーによってのみ用いられ，そして彼が用いている一連のプログラムの作動中のみ存在する．ユーザーがサインオフすればこれらは全部なくなる．またこれらのローカル変数は他のユーザーらが別の端末にロッグインしていても彼らからはアクセスできない．このような“ローカル”という言葉は他のプログラム言語で用いると混乱を起こすことがある．たとえば APL とか，ALGOL や PASCAL などに通じている人々は特に十分注意してこの“ローカル”という言葉を知っておく必要がある．

MUMPS 言語のおもな利益の1つにファイルを共有する点がある．このときには多くのユーザーが用いるのであるから変数を定義する際にすべての人がアクセスできるように，そして誰が自分自身のプログラムを変更してもファイルの変数が変化せず，以前からのままであるように設計しなければならない．このような共用しようと思えばできるファイルを MUMPS はグローバル変数 (global variables) によって満たそうとしている．グローバル変数はローカル変数と同じ性質のものであるが，異なる点は MUMPS システムのディスク記憶装置内にあって，明確に KILL されることがなければ，ユーザーが端末を

離れても，コンピュータ電源を切ってもいつまでも保持されていて，次回にまた複数のユーザーに用いられるという点である．装備によっては，これは非常に大切な機能であるが，ユーザーの種類によってアクセスできるグローバル変数と，できない変数を規定する機能をもたせ，またユーザーによって値を変更できるものとユーザーが読み出すだけの変数の種類を分けたりして，グローバル変数自体に機密保全とデータの保全をはからせる保護機能を備えている．もっとも一般的なグローバル変数の格納場所は大量格納装置としてのディスクなどである．ディスク類の装置そのものの物理的な特性が，変数へのアクセスの仕方に影響を与えているが，論理的な見地からは常にグローバル変数の性格は同じように扱われる．装置に段々と慣れ，共有のグローバル ファイル (global file) をつくろうとする場合に，段々と装置特有のグローバル ファイルの特徴を把握してゆくことになり，プログラミングに際して有効なファイルの利用法を習得するはずである．

グローバル変数は上向き矢印（^）を変数名の直前に付けていることで識別される．それ以外の面ではグローバル変数はすべて第 10 章でみたローカル変数と同じようにしてつくられる．すなわち，添字付きや添字なしや，添字付きの場合はその添字の数に制限のないこと（行の長さの上での制限以内ならば）など，ローカル変数とまったく同様である．一般的にはグローバル変数はローカル変数と互換性があり，ローカル変数の前に上向き矢印（^）をつけることによってただちにグローバル変数として扱われることになる．$DATA や $NEXT によって，ローカル変数のときと同様の結果が返され，SET や KILL 命令の規則によって扱われる．

例外としてグローバル変数を用いてはならない3つの場合がある．それは，

1. FOR 命令のインデックス変数としてグローバル変数を用いてはならない．
2. READ 命令の引数としてグローバル変数を用いてはならない．
3. KILL 命令の引数で括弧の中にグローバル変数を入れてはならない．

グローバル変数の略式参照

MUMPS 言語はさらに速記術的用法をグローバル変数のために備えている．この用法は略式構文（naked syntax）といわれるもので，添字の並びを部分的に括弧内に入れるだけであるが，その添字を完全な構成で変数の添字にもどすには，直前に参照されたグローバル変数の最後の添字レベルに略式グローバル変数の第1添字を入れたものとなる．たとえば，

SET ^A(1, 2, 3)=3, ^A(1, 2, 4)=4, ^A(1, 2, 5)=5

は，略式構文によれば，

SET ^A(1, 2, 3)=3, ^(4)=4, ^(5)=5

となる．この例では第2，第3の変数の値の割り付けに，第1の変数の最後の添字レベルのみを用いているが，添字付きグローバル変数として完全なものとするには変数名のほかに，第1添字と第2添字を，最初の変数のそれらと同じものにして考えねばならない．完全参照形（full syntax）が1つあれば，そのあと“略式状態”として同一グローバル名で，添字がはじめから最終添字の1つ前の添字までのものを想定することは可能である．“略式グローバル参照”の効果は上向き矢印（^）のあとに，その前に参照されたグローバル変数名をつけて，括弧の中に直前に参照されたそのグローバル変数の最後の添字までの正式の添字をすべて解釈させるものである．略式グローバル変数をあらわすためには常にグローバル名を省略して上向き矢印（^）の次に括弧をつける．ゆえに ^(4)という上の例の中の略式グローバル参照（naked global reference）は MUMPS のインタープリタが ^A(1, 2, 4) として解釈することになる．

略式グローバル参照は多重の添字を用いてもよく，次の命令行，

SET ^A(1, 2)=5, ^(2, 3)=13

を実行するとき，^A(1, 2, 3) の中に 13 が設定されることになる．

略式グローバル参照は，多く同一グローバル名が集まって，添字をはじめから何度も繰り返す手間をはぶき，プログラムを簡潔にするために用いられる．また略式グローバル参照は相対的にグローバル変数値を入れることを許しなが

らも，一方では間接指定（第14章参照）をこれに対して用いられないようにしている．装備によっては，多くのグローバル変数を扱うときの効率をこれによって機械的に助けられているものもある．この略式グローバル参照の利用は，ややだまされやすい面をもっているので注意深く用いるのがよい．特に記憶すべきは，略式グローバル参照は，直前に実行されたグローバル変数によって定義されるのであって，プログラムの中でその直前に出てくるグローバル変数によって，支配されるとは限らないことである．

次のコードを考えてみよう．ここで ^A(I) というグローバル変数があるかどうかをしらべて，それに値が存在するならば，^A(I, J)＝J というグローバル変数をつくって J の 1 から 100 までをつくろうとしたときに，次の例はうまくゆくようにみえるであろう．

```
IF  DATA(^A(I))  FOR  J=1:1:100  SET  ^(I,J)=J
```

しかし実際には，これを実行したときにはグローバル ファイルは 101 レベルの階層をもつことになる．なぜならば略式グローバル構文の 1 回実行ごとに，1つずつ添字の数が増してゆくからである．できあがったグローバルの内容は

```
^A(I, 1)=1
^A(I, I, 2)=2
^A(I, I, I, 3)=3
^A(I, I, I, I, 4)=4
```

などとなってゆき，上で意図した作業を希望どおりにおこなわせるには，

```
IF  $DATA(^A(I))  SET  ^(I, 1)=1 FOR  J=2:1:100  SET  ^(J)=J
```

とすべきである．もう 1 つの例をみてみよう．次の命令行，

```
SET  ^A(1)=1,^(2)=^(3,4)
```

の実行に際して，予期されるであろうように ^A(3,4) の値が ^A(2) に入るのでなく，正しい結果は ^A(3,4) の値を ^A(3,2) という変数をつくって入れることになる．SET 命令では割り付け演算子（＝）の右側にくるものがまず評価され（この際 ^(3,4) という略式グローバル参照が評価され）てから，左側の変数に割り付けがなされるのであることを記憶しておく必要がある．

略式グローバル参照は，一度もグローバル参照がなされていない，またはグローバル変数が定義されていないときにおこなうとエラーとなる．略式状態が定義されていない，エラーとなるのは次の場合である：

1. ユーザーがコンピュータにサイン オンするとき，すなわち一度もグローバル参照がなされていないとき．
2. 1回でも添字なしのグローバル変数が参照された直後，すなわち添字のレベルを指示しえないグローバル参照の直後．
3. $DATA か $NEXT のいずれかが，添字付きグローバルを参照して，その最後におこなったものが定義されていないときの直後（すなわち $DATA が0の値を返したとき）．例として，

   $DATA (^A(1, 2, 3, 4))

   で，$DATA (^A(1, 2, 3))=0 となれば略式状態は定義されないことになる．

LOCK 命令

多人数で仕事をする環境では，特に1つの仕事を対象としているときには各人の仕事を同期させてやる必要がある．さもないと共用データ ファイル内から時間的なずれによる不一致や誤りが発生されることがある．たとえば数種の仕事から1つのリストにトランザクションをつけ加えている場合に，^L というグローバル配列に向ってそれがなされているとする．このグローバルは次第に膨張してゆくが，その膨張の仕方をみる方法として，データが入るたびにもっとも新しいデータ入力の鍵を ^L(0) という指標内にもつこととして，^L(0) の値が10であれば最終的に入ったグローバル変数は ^L(10) であるようにしてみると，各ユーザーからグローバルに書き込む命令文を次の命令行のあとおこなうことを想定できる．

```
SET  INX=^L(0)+1, ^L(0)=INX, ^L(INX)=DATA
```

この命令行が各ユーザーから共有ファイルに書き込むどのデータにも用いられたとしても，まず最初のユーザーのプログラムが SET INX=^L(0)+1

のところで中断されて他のユーザーのプログラム領域が動き出すことは，MUMPS のタイム シェアリング方式ではありうることである．そして，そのときに第2のユーザーがこの命令行を実行したとすると，最初のユーザーのプログラムが再開されて命令行の最後まで実行されたとき，この2人とも同一の INX をもっているので，第2のユーザーから入ったトランザクションは消え去ってしまう（実際は最初に ^L(INX) の値として格納された値が消え去る）ことになる．同期されていない他のユーザーによる仕事は上のような問題をかかえている．この問題は1つのデータ ベース内で多くの作業経過が関連する節 (node)訳注) を変更してゆくような場合に必ず付き物となる．このデータ ベース内の節にデータが入っていてその節がポインターをもっている場合上の例のような不一致や誤りが生ずる．

上の例でみた命令行の一連のコードの実行はその通りに一度に実行されないと致命的な結果を生ずるので，その一連の命令行を，一時にただ1人のユーザーしか利用していないような方法を導入する必要がある．LOCK 命令はこの危険な一連のコードに境界をつけ，この間の一連のコードが実行されるまで他のユーザーの同様に危険なコードが実行されるのを防ぐ機能をいとなむ．

LOCK 命令は論理的な1つのファイル構造を一時的に占有することを指令する．しかしながら LOCK 命令の引数となるファイル名と実際の変数名ないしそのデータ ベースとの間に参照やデータのうごきに関する何の連動もないことが特徴であり，LOCK ABC とは単に ABC という名前の排他的占有権を要求しているにすぎない．このとき他のユーザーのプログラムが LOCK ABC を実行しても前に占有権をもったものがそれを放棄するまで一時的に棚上げされている．引数のない1個の LOCK は現用のファイル占有権のすべてを解く．

LOCK 命令が実行されるたびごとに MUMPS システムはそれまで占有し

訳注) 名前をもつ変数のうち，添字をもつ多元分散配列は階層状の樹枝（トリー）構造として，①根（登録レベル）から分岐する樹枝と，②これらによって上下に連結される変数名とその属性および値を入れる記憶域をもつ．①の樹枝をポインターとよび②の記憶域を節（ノード）とよぶ．

ていたファイル名をいったん放棄する．もし LOCK に引数があるときには次いでこの名前の占有権をもつ．この作動順序は，このユーザーがひょっとしてすでに占有しているファイル名の占有権を待っているような可能性（デッドロック状態）をなくする．

以上の例でみたような問題を解決するために，われわれはその危険な一連のコード（危険区間）を LOCK 命令で両側から括弧に包み，かりにわれわれの対象とする資源を LIST とよぶならば，

LOCK　LIST　SET　INX=^L(0)+1, ^L(0)=INX, ^L(INX)
=DATA　LOCK

と書くことでこのリストを LOCK するコードが諸ユーザーのプログラムから発生するのを同期させることができる．もちろん同期すべき他のプログラムの危険区間（critical region）も LOCK で同じように括弧で包まねばならない．

LOCK　ABC は ABC が占有されるまで実行を棚上げして待っている．このときいつまでも棚上げが続くことがありうるので，ある一定の期限をもつ時間制限構文を引数に付け，たとえば LOCK　ABC:5 という形で書くことができる．この形は第 11 章で解説した他の命令語と同じで，5 秒以上の実行の棚上げを防ぐ．この時間以内に占有権が得られない場合，$TEST の 0（偽）という値をもったまま，次の実行に移る．占有権が得られた場合 $TEST は 1（真）をとる．時間制限が指定されていなければ $T は影響をうけない．

注意すべき点は，LOCK　A,B とすると A と B の両方とも一緒に占有権が得られるということは起こらないことである．前述したようにすべて LOCK はまずそれまでの占有権を全部放棄することであり，LOCK　A,B とは LOCK　A　LOCK　B と同じであるからこの場合 B のみが LOCK されることになる．1 個以上の占有権を同時に得たいときには LOCK　(A,B) という形を用いる．この場合並記したすべての資源の占有権が一時に得られねばならず，MUMPS はたとえば A の占有権を得て次いで B の占有権が得られるまで待つということをしない．こういうときに，A と B の両方とも占有権が同時に得られないなら，両方とも占有権を確保できない状態となる．

LOCKの用い方が隘路をつくってデッド ロックを起こすことを避けるために，レベルの異なる括弧をつけることがある．たとえば病歴のように1人の患者のファイルがかえられることを避けようとするとき，全患者のファイルが1つの名前になっていれば，一時に1人の患者のデータしか入れられないことになる．患者ファイルの名でロックをかけることになるからである．この際にファイル名に患者特有の鍵をつけておけば別々の患者についてなら同時にデータを入れることができよう．一方で他の機能，たとえば，新しい患者の名前を登録したり，ファイル全体のインデックスをかえるような行為は全患者に共通なデータをかえてしまうことがある．このような操作や他のシステム全体に変化を及ぼす操作をおこなう場合には，それ自体がすべての患者でデータを更新する作業を排除しておくような危険区間を設けておかねばならない．このように階層的な保護をするためには LOCK の配列形式を用いる．もし次のような書き方をしたとすると，

LOCK ^PAT(ID1)

患者識別番号 ID1 をもつ患者のレコードを更新するための危険区間を設定できることになる．他のユーザーは別の患者のレコードを更新する際に，

LOCK ^PAT(ID2)

と書いてその患者だけのレコードを保護することができる．全患者についての占有権をもつには，

LOCK ^PAT

と書けばこれで，下位のすべてのレコードを占有していることになる．LOCK ^PAT が実行されたあと，システム全般の更新をおこなうことができる．この間は，LOCK ^PAT(ID) の形のプログラムに占有権は与えられない．これらが ^PAT の下位樹枝に存在するからである．MUMPS のこのような階層的な LOCK の使い方で，非常に複雑な同期とりを明瞭で簡潔なものにできる．

上の例では LOCK 命令をグローバル変数について，それらを部分的にも全体的にも重複した更新に対して保護されるかを示したが，ここで再び LOCK が，何らグローバルの物理的構造やデータ ファイルに対する直接的な保護作

用をもつものではないことを銘記すべきである．この危険区間の同期とりは純粋にプログラマの慣習的な方法としてみられるべきである．システム管理者はある機種を扱うときにこの LOCK 命令が信頼できるかどうかをしらべておく必要がある．

要　約

この章では MUMPS システムの多くのユーザーに共有され，しかも長期間 MUMPS システムに保有されるグローバル変数の概念を学んだ．これとちがってローカル変数は個々のユーザーが MUMPS と対話形式でやりとりする間だけ存在し，複数のユーザーに共有されることがなく，ユーザーがシステムからサイン オフすれば消えてしまうものである．

MUMPS 言語の主要な力量はグローバル変数をつくる機能にあり，多くの面でこの共有される変数は他のシステムでいうデータ ベース システムと類似点をもっている．これらの間の差異について2つの大きな面を述べるならば，MUMPS ではこれらの変数を用いる前に宣言しておく必要のない点と，第10章で述べたようにこれらのグローバル変数が多元分散配列であるという点である．これらの特徴が，MUMPS に柔軟で可変自在な性格を与えている．しかし，このことがユーザーをしてこの特徴を利用しグローバル変数を自由につくり出したり変更したりする特権をらん用させる危険を，プログラマのレベルでもシステムのレベルでも，しっかりした共有データ ベースの効力に対して常にはらませているということをよく銘記しておくべきである．

第 13 章

標準 MUMPS の特殊な能力

前章までに標準 MUMPS の機能についてはほとんど学んだのであるが，もう少し言語要素について前章までに含まれていなかった機能が残っている．この章ではそれらの機能――パターン照合，$HOROLOG 変数，および $TEXT 関数――について述べる．これらは MUMPS のルーチンを編集したり実行する際にさらに役立つ道具である．

パターン照合(pattern match)

MUMPS は文字列関係演算子としてもう１つパターン照合演算子 (pattern match operator) をもっている．これは前章までに解説した演算子よりも複雑なものであり，多くのプログラムが比較的このパターン照合機能を必要としないという理由でまだ解説していなかった．

場合によっては入力される文字列が，患者氏名に必要な形式をとっているかどうかを検定したり，在庫品コードの書式，日付の書式，その他コード化された事物のコード書式に合致しているかどうかをしらべてから真の入力データとして扱うことが望ましいことがある．たとえば，日付の書式を "MM/DD/YY" という "/" を含んだ書式で数値を入力するように規定されているシステムでは，その入力数値の書式がこの書式に一致しているかしらべたい場合がある．パターン照合演算子は入力される文字列を１字１字の位置について検定するプログラム技術を不要にする．データ文字列を検定するにはパターンという特殊なデータ要素に対比した評価法でなされる．

VAR？パターン

は，VAR という変数がパターンと "マッチ" するか，換言すればパターンで指定した形式をもっているかによって，真か偽かの値をもつ．同様に，

VAR'?パターン

は，VAR という変数がパターンに一致していないときに真という値をもつ.

この概念の理解をすすめるためには，パターンを定義するコードの種々を紹介しなければならない．パターンとは一連のパターン要素の集合であり，その要素は，複数指定子のついたパターン コード (pattern code) であるか，または定文字列である．ANSI 標準では7種の基本となるコードがあり，それらでパターンを定義する[訳注]．それらは，

A……大文字26個と小文字26個のアルファベットをあらわす

C……33個の ASCII 制御文字 (DEL を含む) をあらわす

E……全部の ASCII 文字セットをあらわす

L……小文字26個のアルファベットをあらわす

N……10個の数字をあらわす

P……33個の句読文字をあらわす

U……26個の大文字アルファベットをあらわす

以上のほかに特別な文字ないし文字列はそれらを引用符号（“　”）の間に挟んであらわす.

これらの定義を用いて日付の書式を MM/DD/YY としたいときに，そのパターンは，

2N1“/”2N1“/”2N

であることがわかろう.

このパターン書式は3つの 2N という要素が2つの“/”で分離されていることを示している．“/”は 2N の間で1回しか起こらないことも1という数字で示される．在庫番号など“12-345”という形式をとる場合，それを検定するパターンは，

訳注) 日本 MUG の標準カナ MUMPS 委は MDC に対してカナ文字用のパターン コードとして4個のアルファベット文字を確保するよう要請していたが，これは 1979. 10. 3. (第19回) および 1980. 5. 13. (第20回) の MDC で，「非 ASCII 文字装備では，その非 ASCII (例，カナ) 文字範囲内でその文字との 'patcode's をつくることが許される」という ANSI 標準用の1文節が採決される結果になった.

2N1"-"3N　　パターン

2N1"-"　3N　　これら3つがパターン要素

"12" "-"　"345"　　部分文字列（副文字列）としたパターン要素

個々のパターンコードは ASCII 文字セットに関係があるので，演算子"?"の柔軟可変性を増すために混合形のパターン コードも指示することができる．例を示せば，

AP— これはアルファベット文字か句読文字かのどちらかであるか(PAと書いてもよい）を選ぶ

LN— 小文字のアルファベットであるか数字であるかを選ぶ (NL と書いてもよい)

パターン コードに合致するものは何回起こるかを整数で示すか，または何回あってもよいが，後者ではピリオド（．）をパターン コードの前に置く．もしも不定回数をあらわす「．」が置かれた場合，マッチされる文字列（まったくない場合も含めて）がいくつ存在してもそれはマッチ演算子（?）を満足させるものとなる．不定回数の記号（．）は適当な混合形の入力文字列を探索するときにパターン照合操作が非常に便利なものとなる．

次の例をみてみよう：

パターン照合指示コード	意味
X?1"ME".E	変数Xの値が"ME"という部分文字列ではじまるがあとはどういう文字がきてもよい
X?.E1"ME".E	変数Xの値が中に"ME"という部分文字列をもっていること（これは X ["ME" と書いてもよい)
X?.E1"ME"	変数Xの値が最後に1個の"ME"という部分文字列で終る

患者の名前を姓ではじめ，コンマをつけ次いで個人名を書く形式の場合は，

PAT?1A.A1","1A.A

となるであろう．これで姓も名も少なくとも1文字のアルファベットではじま

ることを指定している．そのあとは不定回数のアルファ文字が大文字であろうと小文字であろうと続くのが許されることになる．これらの間にはコンマが挟まれていなければならない．必ずしも必要でないとしても，コンマのあとにスペースがいくつあってもなくてもよいという場合には，

RAT?1A.A1“,”.“ ”1A.A

としてどんなコンマのあとの書式もわかることができる．他の例を示すために，ここで添字なしの変数名の形式を1つチェックしてみよう（変数名の構文規則は第2章に解説してある）．

NAM?1“%”.NA!(NAM?.1A.NA)

でその構文をチェックできる．ここではアルファベット文字か1個の%が第1文字として許され，あとはアルファベット文字か数字でよいという構文のチェックができる．

$HOROLOG 特殊変数

どんなコンピュータ プログラムでも，その日の日付が出てくれば，さらに時刻までもコンピュータから出てくるようならば，利用価値は高まる．たとえば請求書やトランザクション，その他多くの報告書などの形式は通常日付を書いている．また，1日の間に数回トランザクションが起こる場合にはその日の何時にそれをおこなったかを明記しておく必要もあろう．プログラマは彼のプログラムの一部分の実行にどれだけ時間がかかるか，または1つの仕事がどれだけの時間をくうかをしらべたいことがある．これはシステムの性能をテストする場合にも，またプログラムの効率の良否をみる場合にもなされる．多くのコンピュータでは，“外部時計”がついておりこのような要求を満たすものがあって，その日の時刻を常にもっている．これらの多くは秒の単位にもっとも近い数字で表現される．外部時計をもっている MUMPS 装備は必ず $HOROLOG 特殊変数をもっている．これでユーザーがプログラムの中で日付と時刻の情報を得ることを可能にしている．

$HOROLOG はコンマで分けられた2個の数値を格納している特殊変数で

あり，その形は “D, S”，つまり D が日付で以下に述べる式であらわし，S はこの変数を参照した時点のその日の朝からの秒数である．はじめの部分のDをどの日にもあてはまる（過去の日付も必要となるので）ようにするために，1841 年の 1 月 1 日を 1 としたそれ以後の日数を入れている．MUMPS 開発委員会は 1940 年の終りから今日までの日数があればほとんどの実際的業務に間に合うと考えている．1977 年のクリスマスは 50034 という数字をとることになる．これは 4 年ごとに 1 回閏年があるので 1461 日が経過することを考えてみるとよい．そして 1900 年がもう 1 日多い閏年であったことも知るとよい．

ここで，以上のことを考えて，常にプログラムを作成するのは非常に不便であり，以下に $HOROLOG の中の D 値から本来の日付の書式をとり出すルーチンを書いておく．これは「標準 MUMPS 言語マニュアル」[訳注1] の 101 頁から引用したものである．

```
SET  H=$HOROLOG>21608+$HOROLOG+1460, LEAPYRS
   =H\1461, Y=H#1461[訳注2]
SET  YEAR=LEAPYRS*4+1837+(Y\365), DAY=Y#365+1
SET  MO=1 IF Y=1460 SET DAY=366, YEAR=YEAR-1
FOR  I=31, YEAR#4=0+28,31,30,31,30,31,31,30,31,30,31
   QUIT:DAY'>I SET DAY=DAY-I, MO=MO+1
WRITE  MO\10, MO#10, "/", DAY\10, DAY#10, "/", YEAR
```

このルーチンによって日付が “MM/DD/YY” の数字表現形として書き出される．ここでこのルーチンの動き方を少ししらべてみると，第 1 行では日付が 1900 年以後の場合は 1 日を加え，その後 1461 日という閏年を含む 4 年間が完全に何回含まれるかを計算し，その周期完了後今日までどれだけ余りがあるかを数えている．第 2 行では閏年を含むはじめの年（1837）に閏年の回数を 4 倍し，残りの年数を加え，その後本年になってからの日数を出している．第 3，4 行は毎月の日数を，閏年の場合を含めて前の行で出した日数に満ちるまで加

訳注 1）　MDC 編，若井一朗ほか訳，コロナ社，1977.

訳注 2）　LEAPYRS: Leap Years（うるう年）をちぢめた変数名.

```
%UDQ   ; ZIM, YS; $H  TO  DD-MMM-YY
       ; ALGORITHM  FROM  IGIELNIK  &  MDC  REFERENCE  MANUAL
       S  %UD=$P($H, ",", 1)
U 2    Q : %UD<0   S  %H=%UD>21608+%UD+1460, %L=%H\1461, %YR=%H#1461
       S  %Y=%L*4+1837+(%YR\365), %D=%YR#365+1
       S  %M=1  I  %YR=1460   S  %D=366, %Y=%Y-1
       F  %I=31, %Y#4=0+28-'(Y#100), 31, 30, 31, 30, 31, 31, 30, 31, 30, 31   Q : %D'>%I    S  %D=%D-%I, %M=%M+1
       S  %D=$E(100+%D, 2, 3), %Y=$E(%Y#100+100, 2, 3)
       S  %M=$E("JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC", %M*3-2, %M*3)
       S  %UD=%D_"-"_%M_"-"_%Y  K  %Y, %M, %D, %I, %YR, %L, %H  Q


%UTQ   ; ZIM;GET  TIME (HH:MM:SS)
       ;
       S  %S=$P($H, ",", 2)
U 2    S  %H=%S\3600, %M=%S#3600\60, %S=%S#60
       S  %UT=(%H\10)_(%H#10)_":"_(%M\10)_(%M#10)_":"_(%S\10)_(%S#10)
       K  %H, %M, %S  Q
```

図 13·1 (邦訳補遺) 日付を DD-MMM-YY の書式でローカル変数% UD に，時刻を HH:MM:SS の書式でローカル変数 %UT に設定する 2 つのユーティリティ，書き出すときはそれぞれ W-%UD, W-%UT とする．それぞれのルーチンのラベルU 2 は病歴その他既存のデータの中に $HOROLOG の値が日数値部分の形や秒数値部分の形でそのまま入っているときに，それらを %UD や %S に設定してから DO U2^%UDQ のような用い方をするのに便利である．

え，月を計算する．最後の行はモジュロ（#）と整数除算（\）を用いて日付を"MM/DD/YY"書式で，1桁の数なら0をその前において書き出させる．このようなプログラムを解析することはMUMPS言語の種々のプログラミング技術を習得するうえできわめてよい助けとなろう．

$HOROLOG 特殊変数の第2部分は文字列の第1部と次のようにして分離してとり出される．

SET TIME=$PIECE ($HOROLOG, ", ", 2)

こうすれば，その日の時間は TIME を 360 で整数除算すればよく，分は TIME/60#60，秒は TIME#60 であらわされる．書き出すとき AM または PM をつけることができるが，くわしくは「標準 MUMPS 言語マニュアル」の 101 頁をみるとよい．

$TEXT 関数

効果的なコンピュータ環境で求められるものの1つとしてプログラムの編集機能がある．言語によってはオペレーティング システム訳注）に編集機能を司る責任をもたせている．この方法ではシステムの編集機能を装備する仕方に大きい差異をともなってくる．MUMPS 言語は，はじめからオペレーティング システムに通訳語とファイル管理システムを部分的に兼ね備えるように設計されているので，この編集機能もいくらか異なった方法がとられている．通常オペレーティング システム用にあつらえられる機能でも通訳語の中に入っていたり，しかも，第 15 章でみることになるがこれらの多くの仕様が言語装備者の判断によって特有なものとなっている．

$TEXT はルーチンの編集用に特に有用な関数である．$TEXT の目的はユーザーに，この関数の引数で指定する1つの命令行を返してくることである

訳注） オペレーティング システム：OS，コンピュータを能率よく働かせる自動制御機構で，単位時間に多重プログラムを処理し，おのおのの応答時間を短縮し，言語通訳のモジュールを操作し，入出力装置の稼動，記憶領域の使用効率，ファイル空間の管理，入出力バッファリング，優先処理などのシステム制御と処理のコンピュータに備わった，ユーザー アプリケーション以外のプログラムをいう．

る．第7章でみたように，命令文は，ラベル名をもってはじめることができ，ルーチンは多くの場合ラベルをもつ命令行が数個あって，その間にラベルのない命令行が挟まっている形をもっている．$TEXT の指示をうける引数はそこにラベル名を入れてもよく，ラベルをもった行からはじめて何行目という指定をしてもよい．たとえば次のようなルーチンがあったとする：

```
ADDRESS ; FNU, 11/13/78
        ; ジュウショ ジョウホウ ノ ニュウリョク プログラム・
          CR デ オワル
        WRITE !,"ナマエ, バンチ, グン・シ・チョウ ソン, ケ
          ン・ユウビン NO.,"
        WRITE !,"シレイドウリ. オワル トキ ハ ナマエ ニ CR
          ヲ イレル."
START   FOR I=1:1 QUIT: $DATA(^NAM(I))=0
        READ !,"ナマエ: ", N IF N="" GOTO FINISH
        SET ^NAM(I, 1)=N
        READ !,"バンチ: ", N SET NAM(I,2)=N
        READ !,"グン・シ・チョウソン:",N SET NAM(I,3)=N
        READ !,"ケン:", N SET NAM(I, 4)=N
        READ !,"ユウビン NO:", N SET NAM(I, 5)=N,
          I=I+1 GOTO START+1
FINISH  WRITE !, I, "メイ ガ イマ グローバル ニ トウロク サ
          レテイル." QUIT
```

以上の多数の命令行のうちの数行を参照したいときを想像してみる．$TEXT 関数を用いて $TEXT の引数に適当なものを選べばどの行でも参照することができる．以下に $TEXT の利用例をあげてみる．

式	結果としてとられる値
$TEXT (START)	START FOR I=1:1 QUIT:$D(^NAM(I))=0

```
$TEXT (START+4)  READ  !, „グン・シ・チョウソン:", N  SET
                         ^NAM (I, 3)=N
$TEXT (+9)       READ  !, "グン・シ・チョウソン: ", N  SET
                         ^NAM (I, 3)=N
$TEXT (FINISH+5)  (空文字列)
```

$TEXT はルーチンの最初の行（最初の行はライン１である）からはじめて整数絶対値によるオフセットを用いることも，ラベル名とそこからはじめる任意のオフセット値によって行を指定してもよい．

ルーチン内のどの命令行をも指定できる能力をもてば，その行のある一部分を指定して，それを別のものと置きかえる編集用ルーチンを書くことは非常に容易なこととなる．$FIND などの他の数種の関数や命令語を一緒に用いるとこの作業はよけい楽になる．

要　約

この章では以前の章に含まれていなかったが，プログラミングと報告書のために有用な数種の言語要素を学んだ．最初，入力文字列の書式のチェックに便利なパターン照合用文字列演算子について解説した．このパターン照合演算子は応用範囲が限られてはいるが，ある状況のもとでは非常に価値ある機能を営む．この演算子の解説で，MUMPS 言語に備わるすべての演算子がカバーされたことになる．次いで $HOROLOG 特殊変数で日付と時刻の情報を報告書作成やプログラム効率の評価のときに用いることのできるような形で検討した．最後に，編集機能の１つとして $TEXT 関数を例示した．次にくる２つの章で，MUMPS 言語のすべての言語要素についての解説が完了することになろう．

第 14 章

間 接 指 定

間接指定（indirection）とは MUMPS プログラマに MUMPS コードを動的につくり出しそれを実行させる能力を与える機構である．これによってプログラマはある制限のもとで，ある式として成り立つものの中へ一時的にその値を挿入しプログラムの実行過程をかえることができる．

MUMPS では間接指定は4つのレベルで起用できる．それは命令語，引数，名前，およびパターンである．この4つのうちもっとも簡単なものは命令語レベルの間接指定である．命令語レベルでの間接レベルは XECUTE 命令を用いておこなわれる．XECUTE 命令につく引数は式として評価される．結果としてもたらされた式はそれが命令行であるかのごとくに解釈されて，そのコードが実行される．たとえばそのような文を書いた場合，

SET　L=“WRITE　!, ““ ジッコウ サレタ”””

これを次のような文で実行にうつすと，

XECUTE　L

行を改めて，“ジッコウ サレタ” という引用符号をともなった文が書き出される．変数 L の値とはここで命令行として扱われ，それが XECUTE　L 命令の部分にプログラムがきた時点で，あたかもそこに挿入されたかのように実行される． XECUTE　L というような命令が，何度も実行を繰り返すようなループの中に置かれた場合，変数Lの値は実行のたびごとに挿入されうるので，結果としての出力はそのつど変化しうる．

XECUTE 命令は複数の引数をとることができ，これらの引数はその順序で評価され実行される．たとえば次のような命令行があるとすると，

XECUTE　L, “S　Y=3.7”, “DO　^ABC”

これらの3つの引数は左から右へと実行され最後にルーチン ABC 全部の実行

で完了する.

XECUTE の引数群はそれぞれの引数に後付け条件を用いて条件指定をおこなうことができる. この後付け条件法は第8章で解説したものである. この技法が有用になるのはそのような場合である. 変数Dの中にデバッグ用の命令語を割り当てて, ただDBという条件が真となったときにのみデバッグ用の命令文が動作するという状態を仮定すれば,

XECUTE A, D:DB, B

は常にAを実行してBを実行し, ただDBが真である場合にのみAとBとの間でDは実行されることになる.

次の3つの間接指定の形式（引数, 名前, およびパターン）は特殊の構文規則を要する. これらの場合には間接指定は間接指定演算子 "@" (indirection operator) によって指定され, そのあとに間接指定式がくる. 言語仕様によると正式には間接指定に用いられる式はどんな式でもよいのでなく式子 (expression atom)[訳注] である必要があることになっている. 簡単にいえば, もし間接指定の選んだ式が1つの変数名や, 1つの定数, 1つの関数でない場合にはそれらを括弧で囲んでおく必要がある. たとえば A－B－C という式は間接指定の文脈で用いてはならず, (A－B－C) としてはじめて式子になりうる. 同じ理由で選ばれる式をみるとき, A, Q(3), ^B, "S X=3", $PIECE(S, D, M) はすべて有効な式子として間接指定で扱いうる.

引数レベルの間接指定は1つの式子の値が命令語の1つかそれ以上の完全形の引数 (complete arguments) を定義する能力をつくり出す. この型式の間接指定は間接指定演算子@を命令語の引数の第1位に置く. この演算子が MUMPS に検出されると@に続く式子が評価されて, あたかもそれが命令語の1個ないしそれ以上の完全な引数であるかのように実行する. これを例示するために, 次のように変数の割り付けがなされていると仮定し,

訳注） 式子は式のもっとも単純な形で, 値をもちうる最少単位であり, 二項演算子などとより複雑な式をつくる. 式子は特殊変数, 関数, 定文字列, 変数, '真偽値式子, ＋か－を前においた数式子などである.

X1＝"A＝3"　X2＝"B＝6, Y＝Z"

そして，SET　@X1, @X2, Q＝"ABC"という命令行を実行した場合には A に 3 という値が，B に 6 という値が，Y に現在の Z のもつ値が，そして Q に文字列 ABC が割り付けられることになる．しかしながら，SET　@X1+Z という命令行は規則違反になる．その理由は間接指定は SET 命令の指示する完全な引数形を満足させるものでなければならないからである．

引数レベルの間接指定は FOR 命令を除いて引数をとることのできる命令語のすべてに対して利用できる．FOR では引数レベルの間接指定は許されない．

名前レベルの間接指定は MUMPS 言語の若干の制限された状況の中でのみ許される．引数レベルの間接指定と同様に名前レベルの間接指定は演算子@ではじまる式子で表現される．式子が評価されて，間接指定のあるその場所に置きかえて実行される．式子の価は MUMPS でいう名前の構文規則に合致するものでなければならない．名前レベルの間接指定が許される場合は次のとおりである．

1)　ローカル変数でもグローバル変数でも，変数名として許される場所．たとえば，X＝"A" であるとすると，命令行 SET　@X＝4 は A＝4 と設定するのと同じ結果になる．

2)　命令語または関数でおこなうベルを呼びうる場所で，もし X＝"LINE1" ならば，GOTO　@X^ROU は ROU というルーチンの LINE 1 のラベルへ飛ぶ命令を実行する．

3)　命令語でルーチン名が参照される場所．たとえば，DO　LAB^@X，として X が，X＝"RTN" とか何かのルーチン名を別に指定されている場合，そのルーチン名を呼び出して中の LAB という行から実行が開始する．

4)　LOCK 命令に名前参照を可能にする場合．たとえば，LOCK　@X として，X が，X＝"^A(1, 2)" という指定がなされている場合．

パターン レベルでの間接指定は名前 レベル の間接指定に非常によく似ている．しかしながらその用い方は非常に特異的で，間接指定演算子がパターン照

合演算子“?”のすぐあとに続いていなければならず，そして間接指定引数（式子）が規定にある MUMPS パターンの値に評価できるものでなければならない．下記の例で A＝“1A”, B＝“1N. N1A”，および C＝“@A”とすると，

プログラム コード	結果のコード
X?@A	X?1A
X?@(A_B)	X?1A1N. N1A
X?@C	X?1A

最後の例は間接指定が入れ子（ネスティング）にできることを示し，1つの間接式を評価すると，もう1つ下に間接指定のレベルがある．この入れ子ができる間接指定の用法はパターン レベル だけに限られたものでなく，命令語，引数，名前レベルの間接指定内でも用いることができる．

要約

間接指定は強力な道具であり，ユーザーがプログラムの働きに柔軟性を与えるために用いる．多くの理由でこの間接指定は注意深く用いほどほどにとどめておかないと，間接指定に依存するような書き方をすればするほどプログラムは解析しにくく，仕様化がむつかしい．これは自らを変化させるコードが，その追跡を困難にするからである．安全なプログラミング技術は，間接指定でないとその仕事がむつかしくなるときにのみ用いること，そしてコメントを付けてこの間接指定があらわれるたびにその意図と範囲をはっきりさせておくことをしいることになろう．

第 15 章

MUMPS の装備特異的な要素

MUMPS の ANSI 標準は一連の高度に装備依存的な命令，関数，変数を含んでいる．この章ではこれらについて短く解説して，このような装備依存要素についての認識を深める助けとしたい．自分の扱う MUMPS システム装備についてこれらの要素の詳細を知るにはメーカーからの仕様を読むか本書の第19章のあとにその要約が挟み込んであればそれを読むとよい．

BREAK 命令，VIEW 命令，および $VIEW 関数

BREAK 命令は標準 MUMPS では，非標準的なプログラミング機能への移行点をつくるために用意されたものである．これが引数なしで用いられた場合は，BREAK 命令の実行はプログラムの実行を次に特定の信号が入力されるまで棚上げする．特定の信号は標準仕様では定義せず，装備に特異的なものとして扱っている．このようにして，BREAK はプログラマが中断したいところでプログラムを“凍結”して，機種に備わる機構によってそのときの条件，変数の状態，などを検査し，その後また実行を再開させることを可能としている．標準 MUMPS は装備に特有な BREAK の引数を許し，その引数の利用法を装備者に定義させている．

VIEW 命令は装備に特異的な引数をとることのできるもう1つの標準命令語である．標準 MUMPS は，VIEW 命令で装備特有の情報を検出する方法を提供している．たとえば，装備者は VIEW 命令でプログラマが記憶域の一部分を“ダンプ”して，データやコードを検出するためのデバッグ用補助を提供する場合がある．

$VIEW 関数は VIEW 命令と同様に，機械固有の情報を検出するために用いられる．この関数の引数は規定されていない．一般的にこの関数は各引数に

従って機械特有の一片の情報を返すようになっている．

VIEW 命令と $VIEW 関数の使用は，これらの操作がシステムをかえるたびに異なるのでそのプログラムを移送不能とする．この理由によって，プログラムの互換性を意図したときはプログラマはこれらの命令や関数の使用を制限すべきである．

$JOB 特殊変数および $STORAGE 特殊変数

一般的に $JOB と $STORAGE 特殊変数は特に小さい MUMPS 機器の場合よく用いられる変数であるといえる．$JOB はコンピュータ内での作業番号 (job number) を整数値でもっている．また，$STORAGE を参照するたびに，これはそのユーザーのために残されているコンピュータ内の記憶域容量を文字数で返してくる．しかしこの数値は命令行などに正確に換算するときの方法は用いる機種の特異性によってことになる．

装備定義による命令語，関数，および特殊変数

これまでの章で学んだように MUMPS は命令語，関数，および特殊変数のすべてはそれらの第1文字のみに省略することができる．SET は S，$FIND は $F，などと省略型を書くことができる．標準 MUMPS は MUMPS システムの命令語や関数，変数にアルファベットの全文字をあてていないが，他のすべての文字は将来用いるために保留している．この保留が MUMPS システムに潜在的な制限を加えている．たとえば，自分の機種に何か特別の命令語を装備しようとするものが，まだ標準 MUMPS に用いられていない命令語をつくろうとしても，この制限によってその選択を妨げられる．新しい命令語や，関数，特殊変数を装備しようとするときのために，それらの第1字目にZの文字を付けて用いようという慣例ができている．Z文字は非標準命令語，非標準関数，非標準特殊変数がこの第1文字に続くことを示すだけであり，あとに続く名前がそれぞれの目的とする作動に適切と思われる独特の定義部となる．たとえば，もし装備者が平方根の関数を用意しようとすれば $ROOT という名

前を付けたいだろうが，$R はすでに $RANDOM という特殊変数で占められており，そのために $ZROOT という名前にすれば，その省略型は $ZR としてあらわされよう．第1文字のZはこの関数が言語装備特異的なものであることをあらわす．これらの場合，省略型では常にZと次の1文字で，他のZ指示と区別がつくようになっていることに注意すべきである．ほとんどの装備がこの拡張法を用いているので，システムに付け加わったこれらの機構をあらかじめ装備の仕様書でよくしらべておくのがよい．この際，これらの拡張機能を用いればプログラムが移植性を失うことを知っておくべきである．

直接モード(direct mode)

今までに学んだ MUMPS の命令語は格納された MUMPS のルーチンの一部として実行されるものであった．この形式の実行を“間接的”または“プログラムされた”モード（program mode）の実行形式であるという．MUMPS の機種のほとんどがこれ以外に“直接的”モードでの実行形式を備えている．それは端末でタイプするごとにその実行がおこなわれる命令語である．こういう機種では，次の命令行をタイプすると，

```
WRITE 1+2+4/2
```

ただちに，

```
3.5
```

という結果が端末にあらわれる．いいかえればこの命令語は“直接的”に実行され，しかも，プログラムの一部として将来用いられることはなく，記憶域に格納する手続もなされていない．こういった直接モードの操作はしばしばルーチンを作成したり，修正したり，格納したりするために用いられる．たとえば，ここに ADMIT という名のルーチンが格納されておれば，これだけを駆動するためにユーザーが，

```
DO ^ADMIT
```

と端末でタイプするのは日常の作業である．この際ユーザーは格納されているコードの間接的実行を，直接モードから要求している．ルーチン ADMIT が

実行を終了すると，システムはこのプログラム モードから再び直接モードにもどり，ユーザーがもっと他の活動を要求できるようにする．

直接モードのもう1つの有益な利用法に，プログラムの虫をとる作業がある．前章で述べた BREAK 命令は，しばしばプログラム中で用いられ，その実行時に処理過程が一時中断して，そのとき直接モードをユーザーに返してユーザーが自由に処理段階での中間結果をチェックするという目的を満たす．この環境が与えられたときにだけユーザーはルーチンの作動のチェックだけでなく，ルーチン内の条件をかえてみて，もう一度間接実行をさせることができる．

直接モードをもつ MUMPS 装備では，ただ直接モードのみ，または直接，プログラム モードの両方に用いうるという操作上の制限をいくつかもっている．これらの制限は標準 MUMPS で決められていず，それぞれの装備の仕様で詳細を知る必要がある．

要　約

以上の章で，ANSI 標準に定義された MUMPS 言語の要素を解説したことになる．また個々の MUMPS 装備が言語に新しい機能を付加するときの方法についても言及した．これ以後の各章では，コンピュータ言語としての MUMPS が進化発達上どのような方向におもむくかについてしらべてみたいし，また MUMPS の利用法で特異的な例をあげてみたい．

第 16 章

MUMPS の 将 来

本入門書では MUMPS を ANSI 標準に定義されたとおりに紹介するように努めた．今では MUMPS は世界中広く利用される言語となりすでに 1200 をこえる施設で用いられている．また多くのコンピュータ装備者やメーカーが標準 MUMPS をそれぞれ 1 機種ないしそれ以上の機種に装備をはじめるまでになっている．

コンピュータ言語は静止状態でとどまっていない．広く用いられるようになるにつれて言語の拡張に対する新しい進歩的な案が出てくる．第 1 章で短く紹介した MUMPS の発祥とその歴史からみても初期の普及時に多くの方言が発生したように，多くのコンピュータ言語が陥りやすい方言化の運命から MUMPS も解放されているわけではない．米国連邦政府の支援によって MUMPS 言語の定義化をしてゆく過程が標準化された MUMPS を生むまでになり，今後も新しい定義が付け加わってゆくことになっている．いったん標準言語仕様が制定されると，そこからスタートしてユーザーの要求を受け入れた新しい言語拡張への機運が出てくることは当然のことである．

現在の標準 MUMPS (1977 年 ANSI) からスタートして，どのような変化が将来みられることになるであろうか？ 標準化の延長上に何か言語の生成をはばむような制限は何もないか？ MUMPS 言語は再び互換性のない方言の集合と化してしまうようなことはないか？ 逆に，ある機能への要望が強まったとき，これらをとり入れて標準 MUMPS をよく統制された方法で進化させる望みはかなえられるだろうか？ これらの問いに対する答えはきわめて明るいものがある．MUMPS 言語の進化の過程の企画はすでにはじまっており，将来の MUMPS の進展に関する一般的な予測をすることも可能となっている．この章ではこれらの問題について，企業や，保健，医療，学術団体などに

よって広く用いられる恒久的な言語として受け入れられるための最終的な課題に向けて解説してゆくことにする.

変化の過程

MUMPSが米国国家規格協会(American National Standard Institute — ANSI)によって標準言語として採用されたとき[訳注], それは採用の代価として限定された環境を与えられた. それはANSIの規則によって採用後5年間標準仕様を有効期間とするということ, そして5年期間の終りにもう一度再評価して前回どおりのまま標準とみなすか, またはそれまでの開発経過を反映させて標準の改訂をおこなうかということにされている. MUMPSユーザーズグループの常置委員会として, 正式にMUMPS開発委員会がおかれ, その委員はその規程を受け入れる人々の集団に対して開放されており, 誰でも参加することができる. ここで現在の標準MUMPSの改訂のための試案が練られて, 1982年の9月以前に, 標準仕様改訂案としてANSIに提出されることになっている.

MUMPS開発委員会(MDC)の, MUMPS開発のための過程で果たす役割は, 可能な仕様追加, 訂正, および仕様の改変を言語仕様書にほどこすことである. そしてすでに原仕様書内の小さい誤りがユーザーに指摘されて訂正され, さらに委員会では今後の有用な言語拡張の多くの可能性についても, 次期改訂以前に十分に評価したのちそれが採用されるように十分な注意を払っている. 現在の標準仕様に対して1982年以前には正式な言語仕様の変更は起こりえない. 言語装備者(language implementor)は自由に拡張をしてみて, その言語的効果について報告することができる. このような経過のあとで1982年にはよくととのったある種の言語拡張が起こるはずである.

MDCの規約は提唱された言語拡張機能を新しく標準仕様書に付け加える前にどういう手続きをとるかを具体的に示している. 変更に関する提案は, 適当な小委員会にまずもちこまれ, ここで通過した議案は全MDC委員会にかけら

訳注) 1977年9月15日.

れ，ここで 1982 年用の仕様書のための新しい改訂操作が加えられる．このような過程が MDC に果たしている責任体制からみて，MUMPS 言語の進化が今後はやや緩慢なものとなることは当然である．しかしながら，すでに次期の標準改訂に挿入される可能性の強いいくつかの機能が検討されつつある．以下にこれらの機能について短い紹介をおこなっておく．

言語の拡張仕様

提案されている言語の拡張案のすべての中で次期の標準仕様の中に織り込まれることが決定しているといってもよい1つの機能がある．それは文字列添字(string subscript) の機能である．文字列添字とは，ASCII の印字される文字の中からとった文字が集まって添字値となる添字のことである．例をあげると，

INVENTORY ("ITEM 1"), ^PATIENT ("SMITH"),
LABTEST (DATE, TIME, "HCT")，など

である[訳注]．この形式は今まで許されている整数値の添字に対して著しい拡張がおこなわれたことを示すであろう．すでに MUMPS 言語がもっていた分散配列概念と自動的序列付けの機能をみれば，敏感なプログラマにはソート機能が文字列の操作をするプログラムでは明らかに必要となってくることが察知されたであろう．広いユーザー層からの要望がよせられ，数種の装備ではすでにこの文字列添字のテストがなされている．この機能が成功裏に動くことの実証があるので MDC ではこの提唱を 1982 年の改訂の中に成案とした．$NEXT は文字列添字に対しては無能である．そのため MDC は $ORDER の仕様をととのえ，同一添字レベルでの次の文字列添字の検出と，これに $NEXT の機能をも兼ね備えるようにして，プログラマには $NEXT よりも $ORDER を用いるようにすすめている．

前掲の例は新しい MUMPS が採用するであろう文字列型の添字を紹介したものであるが，そのほかに現在考慮されつつある変更の中には，ある種の引数

訳注）標準 MUMPS 仕様は文字列添字のおのおの31を文字までは最低もたなければならないとして規定されるはずである．多くの装備は 225 文字までを許しうるようにしている．日本の装備では，ほとんどがカナ文字を添字として許すようになりつつある．

を指定して関数やサブルーチンの中へパスさせてやる方法や，ある種の直接モード操作を定義した方がよい，などがあり，MDC ではこれらのほかにも本書に含むには時期尚早な課題を討議中である．

言語進化の制限

標準 MUMPS が制定されて以来 MUMPS 開発委員会の議事の中で数種の傾向が明らかとなってきた．第1には十分な吟味をせずに今までの言語を改訂しないということがある．どんな討議の場にも必ず，変化の早期実現を希望する派と，わずかないし何も変化を望まない派があるが，MUMPS の場合にはこの両者の均衡は，提唱されれば新しい機能をただちに明白に義務づけるような急激な変化を起こさないほどゆっくりしたものに言語進化と改訂をおちつかせる様相をみせている．このような状態は，装備者に言語の版のメンテナンス問題や，広くみえている将来像の中から十分時間をかけて新しい機能を追加してゆく作業の課題を着実にこなしてゆけるようにするので，言語の進化発展のためにも健全なものとなる．

第2に MUMPS 言語がすべての人々にすべての目的で用いられるようになろうという罠の中におちこむようなことはまず考えられないという点がある．特に MUMPS が科学計算の分野で弱いという点は，そのまま残される問題となろう．本来の MUMPS の設計からみて計算能力を促進するためにデザインされた APL や FORTRAN と計算能力を競うようなことは考えられず，M-UMPS に複雑な数値計算，ベクトルやマトリックス演算，ないし三角函数演算の領域で強化をしてその仕事にあたらせるということをユーザーは欲していない．これらの機能も応用ルーチンとして書かれ，プログラムを共有することもできるが，これらが MUMPS に寄与する度合いは，しかしながら MUMPS の文字列とファイルの操作という現在の標準仕様が特徴とする機能に対して第二義的なものとなろう．

進化への補助としての移送性

すでに本書で MUMPS の魅力ある機構が標準仕様装備機種間での互換性にあることを学んだ．また同時に個々の言語装備者が“Z”を用いて新しい機能を追加し拡張しながら次の標準を求めてゆくことも学んだ．これによって異種の装備をもつユーザーがこれらの新しい拡張についても，特に用いてみて欠くことのできないほどよい“Z”装備である場合，これらも追加してほしいという異口同音の要求を出してくる傾向を生み出す契機をつくることになる．

応用パッケージの移送と交換は，I/O 慣習法の採用をもたらすことになろう．たとえば，ルーチン移送書式などが標準仕様の中に受け入れられるようになるであろう．特異的な編集機能をもつパッケージが仕様書づくりやプログラム修正のためにこれらの慣習法を用いてつくられるようになろう．MUMPS 応用パッケージ ライブラリーの経験がもっと積み重ねられると前節で述べたようなユーザーからの装備の拡張の提唱がきかれるようになるであろう．

技術革新が MUMPS の進歩に及ぼす効果

今日の世界が急速に技術革新をとげている中で将来を占うときに危険を増すものの1つは，それが現在あるものにどんな効果をこの場合 MUMPS に対してどんな効果を及ぼすかという占いである．それでも，今日の市場に受け入れられている変化を背景とした，証拠のある2，3の観察に述べることができるであろう．

MUMPS は元来ミニ コンピュータに装備されたものであった．それ以後 MUMPS は大型中央電算機にも装備されたし，最近ではマイクロ コンピュータに装備しはじめられた訳注)．コンピュータ ハードウエアでの経験が深まるにつれて，言語そのものもまだ変化する可能性もある．

訳注）日本ではすでに日立製作所から HIMEC-10 に，マイクロン社から SBS 8000.A に標準 MUMPS が装備されて市販されている．マイクロ MUMPS は今後 CP/M の OS をもつ Z 80 系の機器に広く装備される傾向があり，これらはミニコンのインテリジェント ターミナルとして強力な働きをすることになろう．

MUMPSの機能を分解して考えることのできる要素として，それが解釈語(インター プリタ)として機能する要素，ファイル操作システムとしての要素，そしてオペレーティング システムとしての要素がある．マイクロ プロセッサのもつモジュール性は MUMPS の解釈語としての要素をグローバル ファイルの操作から分離してしまうことを可能にするだろう．実際に，初期のマイクロ コンピュータへの MUMPS 装備はこのことの確認を終えている．このような初期の観察からいって，“分散型”のデータ ベース構成がある種の応用法では望ましい，ないしは適当であるという仮説には誰もが抵抗なく賛同するところであろう．こうなると機器間の情報交換についてのもっと注意深い配慮がなされることは必至であり，こういう環境でのグローバル ファイル交換やデータ交換用の約定も必要になってこよう．

MUMPSの解釈語としての性質は MUMPS でプログラムを開発し修正することを他の言語の何よりも容易なものにしているが，一方ではある環境ではその処理速度の面で受け入れにくい場合もある．この理由から MUMPS システムから，“コンパイル”されたコードを取り出したいという要求も大きいものとなってきている．このような“コンパイル”版のためには MUMPS が現在もっている間接指定の機能を再検討してはどうかという要求もでてくるであろう．

MUMPS 言語はある種のプログラミング，たとえば科学計算などに向いていない．理想的な解決として考えられるものに，この弱点をもつ MUMPS を多言語環境の機種に装備するということがある[訳注)]．そうすれば，この面の性能は他言語にまかせるという方法が取りうるし，このような試みは限られた数の施設ですでにテストされている．今までの経験からは今後この言語がどんな環境の変化をみるか予断はできないが，やがて経験が積み重なるにつれて何らかの目標がはっきりすることになろう．

訳注) Tandem 16 ノンストップはこのような機種として日本に登場している．

要　約

この章では簡潔に今後 MUMPS がどのような変化をするかについて，いくらかの可能性のある方向を示唆しながらまとめてみた．ここではっきりいえることは，今後言語仕様の変化する速度は緩慢であるということであり，それは ANSI 機構の中で5年に一度の改訂しか許されないことと，またそのために働く MUMPS 開発委員会の結果的に保守的な性格にもよっている．どのような改変の提唱がなされても，実際のフィールド テストを含む十分な検討を加えたのちでないと，採用されることにはなっていないからである．

情報処理とマネジメントの把握力に示す人類の成長性からみても，どのコンピュータ言語でもそのまま停滞してしまうとは考えられない．しかしながら MUMPS のデザイナー達はこの進化が秩序立ったものであり，コンピュータ科学全体の受け入れられる変化とともに進むことを期待している．

第 17 章

応用例　文脈内キーワード

はじめに

この入門書はこれまでの章で，読者に MUMPS 言語についてその能力や限界，そして他の言語と区別されるべき機構について紹介してきた.

この章での目的はプログラム例を完全に出して，入ロルーチン，計算，出力，データ セットのサンプルをみせることにある．この応用プログラムは，元来もっと規模の大きいものとして MUMPS ユーザーズ グループ (MUG) からパッケージとして入手することのできるものである．このパッケージから取ったこのサブセットは，Key Word In Context(KWIC) による参考文献整理法である．KWIC のインデックス法がコンピュータによる文献図書管理法にすぐれているのは，それが文献図書をエキスパートによる科目分類作業を経ないで主題インデックスをつくるのにすぐれているからである．ここで文献図書の題名はそれ自体内容を具体的に表現し抽象的なものではないことと，題名の中に内容を示すキーワードがなければならず，キーワードが書かれたらそれはサブ タイトル でなく本題の中におかれるものであって，科目分類にかわりうる性格をもつものでなければならないという 2 つの約束を前提としている.

KWIC インデックス法で用いる方法は文献図書にある語を，接続詞 (and) 冠詞 (a, the)，前置詞 (from, of) を除いて題名からすべてとって，これらの語を頁の中央に縦にアルファベット順に下へ向って並べ，その前後にこれらの語に前後する部分をそれぞれ付け加え，この文献の参照番号を行の最後に付けておくという方法である．これは図書目録にあるすべての文献を一覧するときに役立つガイドとなる．エキスパートでも 10 分から 60 分の時間を 1 つの文献の科目分類に要するのに比して，課目の中にあらわせない詳細度は犠牲とはな

っても，相当な経済性と検索能率をもたらすものである．

MUMPS は以上のような応用法には理想的な用い方ができる．MUMPS 言語の解釈語的性格それ自体がこのパッケージを開発するのに役立ち，その会話モードは文献ファイルの入力と更新のために役立ち，またその文字列操作機能が文献の題のもつキーワードを選んだりソートしたりするのにそれぞれ理想的なのである．MUMPS 以外の言語で書かれる場合は，MUMPS で必要となる費用に比して，はるかに高価なものとならざるをえない．文献キーワード検索用のパッケージが COBOL で最近市販されたものをみると，60ドル以上の価格である．これをみてもこの章1つに込められる価値は，本書1冊を買うときの価値評定を十分納得させるものがあるといえよう．

KWIC パッケージの設計

文献図書ファイル用の KWIC インデックスをつくり出しメンテナンスするために必要な機能は次にあげるものとなろう：

1. 新しい参考文献を入力する．
2. すでにファイルされている参考文献を編集し更新する．
3. KWIC インデックスをプリントする．
4. 文献図書の全範囲をプリントする．

問題の興味ある面にプログラムを集中するために，ここでは 1. と 3. の項目のみに討論する範囲を限り，ファイルの編集更新機能や図書全貌のプリント機能は多様な種々の技術によって完成されるので，これらの問題は個々の読者のデザインにゆだねることとする．

システム ドライバー ルーチン

KWIC パッケージへ，はじめに述べた形のデータを入力するには，ただ一つのメイン エントリー ルーチンで，ユーザーがそのセクションでおこなう作業を選択させるのがよい．このパッケージで選ぶことのできるモードは：

I＝新しいデータの入力

L＝現在のファイルをリストする

K＝更新した KWIC インデックスをうち出す

Q＝終る

（ここで編集機能のエディター メニューを用意しなかったのは，この部分を読者にまかせるためである.）

KWIC パッケージで各ルーチンへの入口をつくるドライバー（エントリーポイント ドライバー）は次のようになろう.

```
KWICMAIN ; RFW, 10/78
         ; グローバル ファイル @REF ヲ モチイル （@でユーザー ベツノ ファイル ヲ ユルス)
         WRITE !, "コノ パッケージ ハ ブンケントショ ノ リスト ヲ ダシ, ブンミャク チュウ ノ KEY WORD"
         WRITE !, "デ ダイメイ ヲ INDEX ノ カタチ ニ カキダシ, ブンケン ニ ブンケン バンゴウ ヲ"
         WRITE !, "ツケテ サクイン シヤスイ セイリボ ヲ ツクル."!
FILEQ    READ !, "アナタ ノ モチイル ファイル ノ ナマエ ヲ イレ テ クダサイ: ", REF
         SET REF="^"_REF IF '$DATA(@REF) WRITE "ファイル ハ アリマセン" GOTO FILEQ
         WRITE !!, "ツギ ノ ナカ カラ シゴト ヲ エラン デ クダサイ:"
         WRITE !, ?6, "I", ?10, "＝アタラシイ データ ノ ニュウリョク"
         WRITE !, ?6, "L", ?10, "＝ゲンザイ ノ ファイル ヲ リスト スル"
         WRITE !, ?6, "K", ?10, "＝KWIC INDEX ヲ ウチダス"
```

```
        WRITE  !, ?6, "Q", ?10, "＝オワル"
START   READ  !, "シゴト ヲ エラン デ クダサイ：  ", CNTRL
        IF  CNTRL?1 "I". E  DO  BIBENTRY GOTO START
        IF  CNTRL?1 "L". E  DO  BIBLIST GOTO START
        IF  CNTRL?1 "K". E  DO  DWIC GOTO START
        IF  CNTRL?1 "Q". E QUIT
        IF  $LENGTH(CNTRL)=1,  ("ILKQ" '[CNTRL) WRI-
          TE " I, L, K マタ ハ Q デ エラン デ クダサイ"
         GOTO START
        GOTO START
        ; メイン セイギョ ルーチン ノ オワリ
```

データ入力

データ入力で新しいデータを追加したり更新する場合に，ファイルとしては閉じた袋ではなく新しい入力を受け入れ，それまでに存在したデータ ベースと順序をつき合わせることができるものでないといけない．また全部の文献について，第1著者の頭文字でアルファベット順に並べた索引簿をつくりたい．このような目的でつくったファイルは次のような項目をもつだろう：

1. 著者の名前は，第1著者の姓ではじまり，イニシャルが付き，次いでイニシャルにはじまって姓におわる第2著者がくる．たとえば Carlson, L. D. and J. F. Green という具合いにならべるものとする．
2. 刊行の年（4桁の整数).
3. 文献のタイトル.
4. 文献学的項目（このプログラムでは各項目を適当な慣習でチェックする部分を省いている).
5. 全内容をうち出すときの順序をも兼ねる参照番号をプログラムによって発生させる.

これらのデータを入れるルーチンは次のような段階を踏まねばならないであろう：

1. 今回新しいデータを入力するまでに，いくつの文献が登録されているかをたしかめる．
2. 順序よく上記の5項目の情報を受け入れさせる．
3. データ ファイル に新しく記憶場所を生成し，出力順番を前記のようにつくり，情報をあとの利用が可能な形で格納する．

また，入力ルーチンは入力している端からユーザーの意志で終らせることができなければいけない．

以上の文献の列記が入るべきファイルは2つのレベルのグローバルを用い，それに次のような性格が必要となろう：

ファイル名：@REF (間接指定によってユーザーは彼ら自身のファイルを指定できるようになっている．したがってこのプログラムは複数の文献図書整理簿として用いることができる．

要素：

第1レベル：1個の整数で1からはじまり，文献が入力される順番を入れる．

第2レベル

1. 著者 ;
2. 題 ;
3. 刊行の年 ;
4. 掲載誌または出典 ;
5. 参照番号.

次の MUMPS ルーチンは上記の作業をすべておこなう．

```
BIBENTRY; RFW, 10/78
          ; ブンケン ノ レッキ(チョシャ, ダイ, ネン, レッキ, サン
            ショウバンゴウ) ヲ ウケツケ
          ; ソレ ヲ ハイレツ @REF (N,1-5) ニ カクノウ スル.
          WRITE !!, "イマ ツイカ スル ファイル ノ ナマエハ ",
            REF, "  セイリボ デス.",!
```

```
        WRITE !,"ウナガシ ガ デタラ ツギ ノ ショシキ デ
          イレテ クダサイ"
        WRITE !,?5,"チョシャ: チョシャ (セイ, カシラモジ,
          ソレカラ ハ カシラモジ, セイ)"
        WRITE !,?10,"ナンギョウ ニ ワタッテ モ カマイマセ
          ン. CR デ チョシャ ヲ オエマス."
        WRITE !,?10,"チョシャ: ノ ウナガシ デ CR ヲ オスト
          ニュウリョク ハ オワリマス."
        WRITE !?5,"ダイ:ダイ ノ スベテ (ナンギョウ ニ ワタ
          ッテ モ カマイマセン. CR デ オワリマス)."
        WRITE !?5,"カンコウ ノ トシ: (4 ケタ)"
        WRITE !?5,"ケイサイシ: ケイサイシ マタ ハ シュッテ
          ン (フクスウギョウ デ ヨイ.)"
        WRITE !?10,"CR ヲ オスト ケイサイシ ノ オワリ ニ
          ナリマス)"
        WRITE !?5,"サンショウ バンゴウ: リスト ノ ソート
          ニ モチイル バンゴウ.",!
        SET LIMIT=5000 ;ブンケン サイダイ キョヨウリョウ
          (ニソイ デ アル)
        ;
        ;***ゲンザイ ノ グローバル ノ オワリ ヲ ミル ループ***
        SET I=$NEXT(@(REF_"(0)"))-1 SET:I=-2 I
          =LIMIT GOTO:I>0 BIBENTER
        WRITE !,"ファイル ガ イッパイ. マニュアル デ カクチ
          ョウ ホウ ヲ ミヨ."
        ;
        ;***アタラシイ ブンケン ヲ ウケイレル ループ***
BIBENTER READ !,"チョシャ: ",X IF X="" QUIT ;デグチ
```

```
         SET  ^(I, 1)=X
NXTTL    READ  !, "ダイ:  ", TITLE
         SET  ^(2)=^(2)_"  "_TITLE IF TITLE'=""  GOTO
           NXTTL
RDYR     READ  !, "カンコウ ノ トシ: ", Z IF Z'?4N  WRITE
           "4ケタ デ" GOTO RDYR
         SET  ^(3)=Z
NXTCIT   READ  !, "レッキ:  ", CIT
         SET  ^(4)=^(4)_"  "_CIT IF 'CIT=""  GOTO
         NXCIT  READ  !, "REF:  ", Z  SET  ^(5)=Z, I=I-1
           GOTO  BIBENTER
         ;***エントリー ルーチン ノ オワリ
```

KWIC インデックスの書き出し

文脈内キーワード（KWIC）インデックスは，文献の題の中にある各語を検証し，対象としない語を有意義な語から除外し，頁の中ほどにおいた縦線に沿ってキーワードを並べてゆく．この書き出し例はこの章の終りにみられるような形となる．このようなインデックスをつくるには，第1に小さな意味のない語の“辞書”をつくっておく必要がある．これらはソートしても意味がないのでキーワードの対象としない語の集まりである．これらを“ストップ ワード”とよび，ストップワードのグローバル ファイルは次の ルーチン を用いてつくることができる．

```
STOPWORD  ; MJS, 6/76
          ; ゴ ノ ナガサ ニ ヨリ ファイル サレル ストップワー
            ド ノ ジショ ヲ
          ; ツクリ ツイカ ヲ スル ルーチン
SWLOOP    READ  !, "アタラシイ ストップワード  ヲ  イレナサイ
            (オワリ ハ CR):  ", WORD
```

```
        IF  WORD=“”  QUIT
        SET  L=$LENGSH (WORD) IF L>10  WRITE  !,
         “モジスウ ヲ 10 イカデ” GOTO  SWLOOP
        IF  $DATA(^SW(L))  S  ^SW(L)=“”
        SET  ^SW(L)=^SW(L)_“/”_WORD GOTO  SW-
         LOOP
        ; ルーチン オワリ
```

このルーチンが実行されると，SW グローバル ファイルに同じ語長の他の語と結合して新しい語を付け加える．すべての語は間に境界子(／)で分離して語の識別ができるようにしている．

KWIC インデックス用の第2のユーティリティ ルーチンはソート用ルーチンである．このユーティリティはシステムの中の構成因子として入っていてもよいものであるが，そのような汎用のルーチンがあると考えられないのでこのパッケージとともに用いうるユーティリティを次のようにつくる．

```
HEAPSORT  ; SLJ, 9/78
          ; サンショウ スル グローバル ハイレツ カラ ローカル
           ハイレツ A(I)
          ; ト B(I) ＝ ソレゾレ キー  ト インデックス ヲ イレ
           ル. N コ ノ ソート ヲ スル.
HPSRT     SET  L=N\2, R=N
          IF  L>1 SET  L=L−1, X=A(L), Y=B(L)DO SIFT
           GOTO HPSRT+1
          IF  R>1 SET  X=A(R), A(R)=A(1), Y=B(R), B
           (R)=B(1), R=R−1
          IF  DO SIFT GOTO HPSRT+2
          QUIT  ; HEAPSORT ルーチン オワリ
SIFT      SET  J=L
          SET  I=J, J=2*I
```

```
          IF  J>R GOTO ENDSIFT
          IF  J<R IF A(J+1)]A(J)  SET J=J+1
          IF  X]A(J) GOTO  ENDSIFT
          SET  A(I)=A(J), B(I)=B(J) GOTO SIFT+1
ENDSIFT   SET  A(I)=X, B(I)=Y
          QUIT
          ; HEAPSORT ルーチン オワリ
```

この "Heap Sort" というルーチンで用いられた技術は Knuth の Art of Computer Programming Vol. 3 p. 145 など数種の教科書で解説されているものである．これは，常用される "bubble" ソート法よりもずっと効率のよいものであり，大量のソートをする際時間の節約ができる．

KWIC インデックスに必要なキーのユーティリティが用意されると，次のルーチンで実際の文脈内キーワードを実現することができる．

```
KWIC  ; MJS & RFW, 10/76
      ; HEAPSORT ト ^SW グローバル ヲ モチイル
      ; ローカル STOPWORD ヘンスウ ヲ ツクル コトカラ ハジ
        メル
      FOR  I=1:1:10  SET:$DATA( SW(I))  SW(I)=^(I)
      SET  CLKS=$JUSTIFY(" ", 80) ; 80 モジ ノ ブランク
        ヲツクル
      ;
      ; ツギ ノ ダイ ヲ トリダシ KWIC ノ ギョウ ヲ ツクル
      SET  I=$NEXT(@REF_"  ")+1, J=0
KWTTL SET  I=I-1,X=REF_"("_I_", 2)",KWIN=@X,KWID
        =^(5) Q:$DATA(KWID)=0
      SET PCOUNT=1, KWLNG=$LENGTH (KWIN), NULLS
        =0, PTR=1, KO=0
      ;
```

```
        ; ダイ ノ WORD ソレ ゾレ ヲ ショリ スル
KWLOOP  SET KWRD=$PIECE (KWIN, "  ", PCOUNT), KWL
          =$LENGTH (KWRD)
        SET PCOUNT=PCOUNT+1
        IF KWRD="" SET NULLS=NULLS+1 IF NULLS>2
          GOTO KWTTL
        IF KWRD<3 GOTO KWLOOP ; 1モジ ト 2モジ ノ
          WORD ハ ムシ スル
        IF KWRD'?1A.E GOTO KWLOOP ; WORD ハ ALPHA
          モジ デ ハジマル コト
        ; ココ デ WORD ガ STOPWORD デ ナイカ ヲ ミル
        IF KWL<10 IF $FIND (SW (KWL), "/", KWRD)>0
          SET KO=KO+1 GOTO KWLOOP
        ; KEYWORD ヲ マンナカ ニ オイテ タテセン ヲ ツクル
        ; L1-L2 ヲ ヒダリ ノ ラン トシ R1-R2 ヲ ミギ ノ ラント
          スル
        SET PTR=$FIND(KWIN, KWRD, PTR), L2=PTR-KWL
          +1, R1=PTR, TB=0, LB=0
        IF L2<1 SET L1=0, L2=0 GOTO SETR2
        IF L2<1 SET L1=0, L2=0 GOTO SETR2
        SET L1=L2-23 IF L1<1 SET L1=1
 SETR2  SET R2=R1+(47-KWL) IF R2<KWLNG SET R2=
          KWLNG+1
        SET LB=24-(L2-L1), TB=(47-KWL)-(R2-R1) IF
          L1>0 SET LB=LB-1
        IF TB>0 SET TB=TB+1
        ;
        ; KEYWORD ヲ 'KW(J) ニ オサメ, KEYWORD ノ ギョウ
```

```
  ヲ ^KWRD(J) = オサメル
        SET  J=J+1, ^KW(J)=KWRD
        SET  ^KWRD(J)=$EXTRACT(BLKS,1,LB)_$EXTRACT
          (KWIN, L1, L2)_KWRD
        SET  ^(J)=^(J)_$EXTRACT (KWIN, R1, R2)_$EXTRA-
          CT (BLKS. 1, TB)
        SET  ^(J)=^(J)_"  "_KWID ;コレデ KEYWORD ノ ギ
          ョウ ハ カンケツ スル
        GOTO  KWLOOP
        ;
        ; ココデ KWIC ノ ギョウ ヲ SORT スル
        FOR  J=1:1  QUIT:$DATA ( KW(J))=0  SET  A(J)=
          ^KW(J), B(J)=J
        SET  N=J-1
        DO  HEAPSORT
        ;
        ; ココデ KWIC INDEX ヲ リスト スル
        SET  TITLE="ファイル  "_REF_ " ノ KWIC インデック
          ス", PAGE=1, LINES=0, I=0
        DO  HEADER
KWRITE  SET  I=I+1  IF  I>N  QUIT
        SET  LINES=LINES+1  IF  LINES>55  DO HEADER
        WRITE  !, KWRD (B(I))  GOTO  KWRITE
HEADER  WRITE  #, ?30, TITLE, ?67, PAGE, !  SET  PAGE
           =PAGE+1, LINES=0
```

応用例

前掲のルーチン群はストップ ワードのつくり方を除いてすべて テキスト で

書いた設計の仕様をすべて踏んでいる．ここで用いる文献例は MUMPS に関する4つの文献であり，本書の読者にとっても有益な一般的読み物となるものである．ここではユーザーが，以上のルーチンと対話するときの応答のすべてを記録した．端末にはカナも英小文字も入出力できるものを想定して，最終的なプリントにはこれらを用いてきれいに仕上げてある．はじめの駆動は直接モードの DO で KWICMAIN を呼んでスタートさせている．

>DO KWICMAIN[訳注)]

コノ パッケージ ハ ブンケントショ ノ リスト ヲ ダシ，ブンミャクチュウ ノ KEYWORD デ ダイメイ ヲ INDEX ノ カタチ ニ カキダシ，ブンケン ニ ブンケン バンゴウ ヲ ツケテ サクイン シヤスイ セイリボ ヲ ツクル．

アナタ ノ モチイル ファイル ノ ナマエ ヲ イレテ クダサイ：MPSLIB

ツギ ノ ナカ カラ シゴト ヲ エランデ クダサイ：

I　=アタラシイ データ ノ ニュウリョク

L　=ゲンザイ ノ ファイル ヲ リスト スル

K　=KWIC INDEX ヲ ウチダス

Q　=オワル

シゴト ヲ エランデ クダサイ：I

イマ ツイカ スル ファイル ハ MPSLIB セイリボ デス

ウナガシ ガ デタラ ツギ ノ ショシキ デ イレテ クダサイ

チョシャ：　チョシャ(セイ，カシラモジ，ソレカラ ハ カシラモジ．セイ) ナンギョウ ニ ワタッテ モ カマイマセン．CR デ チョシャ ヲ オエマス．チョシャ：ノ ウナガシ デ CR ヲ オスト ニュウリョク ハ オワリマス．

ダイ：　ダイ ノ スベテ(ナンギョウ ニ ワタッテ モ カマイマセン．CR デ オワリマス)．

カンコウノトシ：（4 ケタ）

訳注） 端末からユーザーが入力する部分にはアンダーラインがつけてある．

ケイサイシ：ケイサイシ マタ ハ シュッテン (フクスウギョウ デ ヨイ. CR ヲ オスト ケイサイシ ノ オワリ ニ ナリマス).
サンショウ バンゴウ：　リスト ノ ソート ニ モチイル バンゴウ.

チョシャ：　Greenes, R. A.　, A. N. Pappalardo,　C. W. Marble, G. O. Barnett
チョシャ：
ダイ：　Design and Implementation of a
ダイ：　Clinical Data Management System
ダイ：
カンコウ ノ トシ：　1969
ケイサイシ：　COMPUTER AND BIOMEDICAL RESEARCH, 2, pp. 469-485.
ケイサイシ：
サンショウバンゴウ：　G001
チョシャ：　Munnecke, T., R. F. Walters, J. Bowie, C. B. Lazarus,
チョシャ：　D. A.　Bridger
チョシャ：
ダイ：　MUMPS：Characterics and Comparison with other Languages
ダイ：
カンコウノトシ：1976
ケイサイシ：　MEDICAL INFORMATICS, 2：3 pp. 173-196.
ケイサイシ：
サンショウバンゴウ：　M002
チョシャ：　Johnson, S. L., R. F. Walters
チョシャ：
ダイ：　MUMPS Implementation on Microprocessors
ダイ：

カンコウノトシ： 77

4 ケタ デ

カンコウ ノ トシ： 1977

ケイサイシ： MEDICAL INFORMATICS, 2：4 pp. 257-267.

チョシャ： Brigham, C. R., J. D. Halvorsen, J. Zimmeman

チョシャ：

ダイ： QUEST： A Teaching Program Driver

ダイ：

カンコウ ノ トシ： 1976

ケイサイシ： JOUR. COMPUTER BASED INSTRUCTION, 3：2 pp. 42-50.

ケイサイシ：

サンショウバンゴウ： B004

チョシャ： Bowie, J., G. O. Barnett

チョシャ：

ダイ： MUMPS： An Economical and Efficient Time-sharing

ダイ： System for Information Management

ダイ：

カンコウ ノ トシ： 1976

ケイサイシ： COMPUTER PROGRAMS IN BIOMEDICINE 6, 11-22.

ケイサイシ：

サンショウバンゴウ： B005

チョシャ：

シゴト ヲ エランデ クダサイ： ? I, L, K マタ ハ Q デ エランデ クダサイ

シゴト ヲ エランデ クダサイ： K

KWIC INDEX OF MPSLIB FILE 1

MUMPS :	Characteristics and Comparisons with other Languages	M002
Design and Implementation of a	Clinical Data Management System	G001
MUMPS : Characteristics and	Comparisons with other Languages	M002
and Implementation of a Clinical	Data Management System	G001
	Design and Implementation of a Clinical Data Management System	G001
QUEST : A Teaching Program	Driver	B004
MUMPS - An	Economical and Efficient Time-sharing System for Information Management	B005
MUMPS - An Economical and	Efficient Time-sharing System for Information Management	B005
Design and	Implementation of a Clinical Data Management System	G001
MUMPS	Implementation on Microprocessors	J 003
Efficient Time-sharing System for	Information Management	B005
ristics and Comparisons with other	Languages	M002
Implementation of a Clinical Data	Management System	G001
me-sharing system for Information	Management	B005
MUMPS Implementation on	Microprocessors	J 003
	MUMPS : Characteristics and Comparisons with other Languages	M003
	MUMPS Implementations on Microprocessors	J 003
	MUMPS : An Economical and Efficient Time-sharing System for Information	B005
QUEST : A Teaching	Program Driver	B004
	QUEST : A Teaching Program Driver	B004
QUEST : A	Teaching Program Driver	B004
UMPS : An Economical and Efficient	Time-sharing System for Information Management	B005
mical and Efficient Time-sharing	System for Information Management	B005

PLEASE SELECT AN OPTION : Q
>

要 約

この章で１つの応用例を示したが，それはユーザーが入力するとそのデータがどのように処理されるかの過程を再現した．この例は，もっと磨き上げて完璧なものとする余地が残されている．文脈内キーワードはあたかも文献の著者以下すべての文脈を輪にして円筒に巻き付け，頭と尾の間に識別を付したようにして，スペースが円筒にある間はどれだけでも新しい文献を巻き付けてゆく形で，キーワードをさがす処理を容易にしている．この方法では題の第１語でソートされた場合に文献は，その尾がちょうど第１語の前にきている．KWIC INDEX のリストを見やすくするための工夫は，比較的容易に読者によって考えられる練習問題となるだろう．たとえば，頁の端にかかる文字ないし，語群は全部切ってしまうか，両側の欄内でそれぞれ二重，三重に折りたたむかするのも一法である．

第 18 章

応用例　郵送リスト パッケージ

はじめに

読者が MUMPS の応用法を学習するもう1つの例を示すために，郵送リストを処理する応用パッケージをこの章で扱う．送付メンバー表をつくり，それを更新しながら，選んだ相手を全リストから抜き出して郵送リストまたは特別グループについて郵送ラベルを書き出すプログラムを書いてみよう．このパッケージで重要な1つの特徴は，ユーザーの定義する指定子である“フラグ”をプリントさせるものにつけうる機能である．フラグは特異的条件，たとえば特別委員会の委員，会費納入状態，メンバーの種類，などであり，ファイルから報告が必要なとき，これらの特異的なフラグ群に対してブール条件をつくって，適当な形式の出力をさせうる点である．

この種の応用プログラムは， MUMPS 言語では容易に作成して拡張し，利用度の高いものにしてゆくことができる．郵送リストはファイルの維持がダイナミックでなければならず，更新はオンライン形式でなされるべきである．この際に MUMPS の文字列操作と間接指定の機能はフラグ機能をプログラムに装備するために重要なものとなる．どのような企業でも郵送リストを維持することは重要なニーズであるから，これをよく練習することが MUMPS の機能を学ぶことと，業務のニーズを満たすことの一石二鳥を果たせば幸いである．

パッケージの設計

パッケージの全体的構造は，図 18·1 に示すとおりである．

図 18·1 でみるように各プログラム モジュールは用いやすく選択時に融通性をもたせてある．作業オプション ドライバーはパッケージを起動し，5つの作

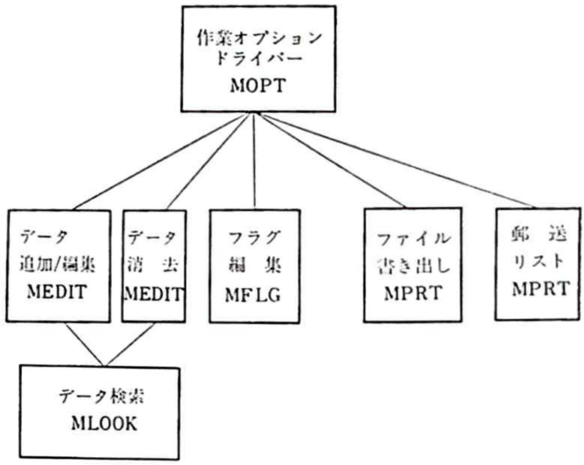


図 18·1　郵送リスト パッケージの構成図

業オプションへのうながしを出力する．追加／編集モジュールと消去モジュールはファイル データのメンテナンス機能を営む．これらの相方ともデータ検索モジュールを用い，特定のデータをさがす．フラグ編集モジュールはフラグ名の索引簿を維持する．ファイルの出力は書き出しモジュールと郵送リストモジュールによっておこなわれる．書き出しモジュールはユーザーの一覧のためにすべてのファイルされたデータを出力する．郵送リスト モジュールは選ばれた名前と住所のリストを郵送ラベルその他の目的のために書き出す．

どんなパッケージの設計に際しても，重要なことはファイル構造とその構成法を定義することである．このパッケージのためには，出力がアルファベット順になされるのが普通であることを考慮して，はじめからファイルをアルファベット順につくっておくのがよい．各個人の名前ないし会社名がファイルの中では第一義的なキーとなる．個人データの検索も姓でおこなわれる．データ検索ルーチンは，入力時にうっかり同一人を二重にファイルしないために入力時に作動するようになっている．

ファイルのアルファベット順の並べ方は，バケット ハッシング という技術でおこなわれる．それは姓のはじめの3文字が数値に変換（ハッシ）され，そ

れをバケット インデックスとする．この１つのバケットに入るすべてのデータはアルファベット順で１個のリスト内で維持される．この手技によると個人データの検索をおこなうときや新しいデータの追加時にオーバーヘッド（データを再度とりにゆき，一時的記憶域をつくりかえること）を少なくする．

グローバル ファイル構成

```
メイン エントリー ファイル名：　^MAIL
^MAIL (HASH, 0) ＝ エントリー数
        HASH=$ASCII (LASTNAME, 1)*100+$ASCII (LASTNAME, 2)*10+$ASC-
        II (LASTNAME, 3)
              , 1 －9： エントリー 1
              ,10－18： エントリー 2
              , ・・・

第0行　＝　名前（姓，名〔中名〕〔，称号〕)……これが１つのエントリー単位を従える
第1行　＝　住所第１行
第2行　＝　住所第２行
第3行　＝　住所第３行
第4行　＝　市
第5行　＝　州
第6行　＝　郵便番号
第7行　＝　電話番号
第8行　＝　フラグ リスト
フラグ索引簿

^FLAG ($ASCII (FLAG))　＝　フラグ状態（任意の文字列）
```

図 18·2　郵送リスト パッケージのグローバル ファイル構造

図 18·2 は郵送リスト パッケージの構成である．データの各行ともユーザーの個別用途に適応させて用いることができる．フラグ リストはアルファ数字による文字列であり，その１つ１つの文字はそれに対応する入力データの状態を記述している．文字とその意味との対応はフラグ索引簿の中に格納されている．フラグ編集モジュールはこのフラグ索引簿の中にフラブ状態を入れるのに用いられ，フラグがデータと連繫するのは追加／編集モジュールの実行中に明らかな方法でなされるが，または指定しないでも暗黙のうちに書き出し機能の中でおこなわれる．

追加／編集モードでは特定データに多くのフラグを付け加え，または消去することができる．郵送リスト操作では，1つ以上のフラグをすべてのエントリーデータに付けてリストに書き出しうる．郵送リストにあるオプションを用いれば，ある群のデータだけを選び，特殊郵送リストを出させることができる．これ以外のファイル データは後日郵送のために書き出すときに，これらのフラグにノット（′）選択条件を付けて，はじめの群の重複書き出しを避けさせることができる．

ルーチンの解説

以下の各項で各モジュールを解説するので，それとプログラム リスティングおよび対話形のサンプルを対照しながら注意深く読むことをすすめる．

作業オプション ドライバー (MOPT)

多くの面でオプション ドライバーの作業は前章での KWICMAIN と似通っている．このモジュールはユーザーに作業のリストから1つ選択させて，そのルーチン テーブルをさがし，適合したものがあればそれを実行させる．ユーザーは，その作業オプションの選択時に何個の文字を入れてもよい．用意されたオプションの1つであるかどうかを決定するために，このドライバーは $FIND, $PIECE および $TEXT を用いている．はじめの文字さえ合致すれば，残りの文字はコンピュータから付け加えられる．他の部分でもそうであるが QUERY 機能として，疑問符号（?）を入れるとそのときに備わっているオプションのすべてがリストされるようになっている．

データ検索モジュール (MLOOK)

データ検索ルーチンは名前を標準的書式（名，中，姓，称号）で読み込むが，それらを内部書式（姓，名，中，称号）に書きかえてメイン ファイルと照合をおこなう．もし適合するデータ照合ができないならば，これと適合しない（閉じられた）名前の一群をコンピュータに書き出してみせるが，これらは同一の

姓をもっている者の名前で，このパッケージでこれらは“閉じられた”群といわれる．非常に多くの名前を登録するような場合には，この方法でなくもっと高度なサウンデックス(SOUNDEX)システムか他の内部コード技術が MUMPS によって開発されているので，それらを利用することもできる．閉じられたデータ群へのポインターは IDX (I) という配列に入っている．これらの中にユーザーの求める名前が見つかったときには，ユーザーはその名前と住所第1行によってそれを取り出すことができ，求める名前のないときには，追加オプションが動き出して，この名前をバケット内のどこに挿入するのがよいかをさがし出す．

データ追加／編集モジュール (MEDIT)

追加／編集モジュールは新しい名前を入力したり，既存ファイルのデータを編集するために用いる．これは汎用 QUERY-応答ドライバーとして設計されており，入力行名が最初にタイプされると，現在の値は括弧内に入れて出力される．たとえば，

CITY 〈BOSTON〉

この括弧内の現在の値がそのままでよい場合ユーザーは単に CR（復帰）を押すだけでよいが，符号“－”を押せば括弧の中が消去され，または置換すべき文字列を書けば括弧の中と置きかわる．ローカル配列Aの中にこれらの応答はいったんたくわえられるが，これらが改訂する内容であるかもう一度一覧させ断定させてから最終的にファイルの中に受け入れられる．

フラグ編集モジュール (MFLG)

状態フラグについての定義はグローバル ファイル ^FLAG（図18·2参照）の中にある．ユーザーは何かのフラグをつくり，その内容をあらわす文章型文字列を割り付ける．フラグ編集モジュールはフラグ名（1個の大文字アルファベットか数字であること）を受け付け，それに対応する説明文字列を要求してくる．フラグがすでにある場合には，その内容が再編集できる．このモジュー

ルでは現在あるフラグと，それらの内容をはじめの“うながし”のあとに星印（*）を入力すれば出力してくれる．

ファイル書き出しと郵送リスト モジュール (MPRT)

ファイル書き出しと郵送リスト モジュールは，2個のルーチンへの入り口をもたせて，これら2つの操作を選択させている．郵送リスト機能に対しては，“条件選択”と“フラグ修正”の2つのパラメータが与えられなければならない．“条件選択”はフラグ名を含んだ論理式として読み込まれ，その後それは MUMPS 言語によるブール式に変換される（ラベル MAKB 以後）．たとえば，もしAとBというフラグをもつすべてのデータのリストが欲しい場合，または，Cというフラグをもっていないすべてのデータを必要とする場合，A & B !' C と書く．この文字列は次のように変化される．

BOOL=F［“A”&（F［“B”）&（F′［“C”）

処理の途中で，どのデータのフラグ ラインにも F は設定されうる．この状態は，サブルーチン NEXT で対象とする氏名のデータを選択するために最終行でテストされる．

すでにわかったことではあるが，データは郵送リスト書き出し処理中にも自動的にフラグが割り付けられている．これらのフラグ（UF）は LIST というラベルのある行で読みとられ，サブルーチン BUFFER の最終部で選択されたデータに付加される．

書き出し機能は郵送リストに非常によく似ているが，それは選択条件ないしフラグ オプションを含まない．書き出し (PRINT) ルーチンは，BOOL の初期値を1とするのでどんなデータも書き出される．BUFFER の最後に書き出し書式と修正すべきフラグの差異が調整処理される．

出力処理そのものは比較的直線的である．ただ複雑な部分はデータを多くの欄に分けて打ち出す点である．いくつの欄を用いるかは COLS というパラメータに入れられる．ラベルの種類がかわるたびに，COLS のパラメータはかえることができる．ルーチンの主要部分は LOOP にある．はじめに，データ

バッファーの BUF が初期化され，BUF は望む欄のおのおのに1組のデータを入れている．次のアルファベット データはサブルーチン NEXT で選ばれる．Sという添字が返され，それがデータ第1行を指示する．郵送リスト機能では，選択された条件を通ったデータのみが返される．もうデータのない場合にはSに負の値が返される．この BUFFER というサブルーチンはデータを配列 BUF の中に形成する．電話番号とフラグ行は書き出し機能のときだけに出力されるが，修正ずみフラグは郵送リストの中だけにつけ加えられる．LOOP +2 の行では，一時に出力を1行ずつ，適当なタブ数を次々と出てくるラベル用に実行しながら出力を打ち出す．

パッケージ ルーチンのリスト

この章では全ルーチンを一度に書き出してみせ，一まとめにして学習するかまたは読者の MUMPS システムに装備するのに便をはかっている．この章の前半の部分で解説が加えられている．このリストでは個々の命令行が本書の行をこえるので何重かにわたるかもしれないが，実装する際には，1個の命令行は 255 文字の長さまで許されることを銘記してほしい．しかし，リストを出力するときに端末の行の制限のために命令行が長すぎるときは，まず MUMPS の端末ならば，自動的に命令行を折り重ねて書き出してくれるものと思ってよい．ときには右端で，折り曲がらずに同一の場所に数十字も印字するものがあるかもしれないが，そのときには装備者に，自動的に復帰改行させて，命令行がすべて打ち出されうるように改良を指示しなければならない．このときに自動発生される復帰改行は，プログラムの命令行の中にはなく，命令行の実行の際に，それを CRLF として解釈するものであってはならない．

MOPT ROUTINE（オプション ドライバー）

```
MOPT  ; ユウソウ リスト オプション ドライバー, JEB, 1/79
      WRITE  !,"ユウソウ リスト パッケージ",!
      SET  ESIZ=9, NL=7, LEN=35, COLS=2  ;データ ノ ギ
```

```
        ョウスウ, プリント ギョウスウ, 1 ギョウ ノナガサ, ラン
        ノカズ
FN      READ ! "サギョウ ハ ?  ", FN  QUIT:FN=""
        ; サギョウ ヲ サガス
        FOR I=0:1  QUIT:$TEXT(TABLE+I)=""  GOTO
          FOUND:$FIND($PIECE($TEXT(TABLE)+I), ";", 2),
          FN)-1=$LENGTH(FN)
        WRITE " ?" GOTO FN ; サギョウ ガ ミツカラナイ
        ;
FOUND   SET X=$TEXT(TABLE+I) WRITE $EXTRACT
        ($PIECE(X, ";", 2), $LENGTH(FN+1, 99), !
        SET FN=$EXTRACT(FN, 1) ; サギョウ ファンクション
          ニ ヨッテ リヨウ サレル
        XECUTE $PIECE(X, ";", 3) GOTO FN
        ;
        ;? ヘ ノ オウトウ ヲ アツカウ
QUES    WRITE ! FOR I=1:1 QUIT:$TEXT (TABLE+I)=""
          WRITE ?10, $PIECE($TEXT(TABLE+I), ";", 2), !
        QUIT
        ;
        ; オプション ドライバー ノ テーブル (プログラム ノ サイ
          ゴ ニ オクコト)
TABLE   ;?; DO QUES
        ; ADD EDIT ENTRY ; DO ^MEDIT
        ; DELETE ENTRY ; DO DEL^MEDIT
        ; FLAG EDIT ; DO ^MFLG
        ; PRINT FILE ; DO PRINT^MPRT
        ; MAILING LIST ; DO LIST^MPRT
```

MEDIT ROUTINE (データ ツイカ/ヘンシュウ/ショーキョ ルーチン)

```
MEDIT ; メイン ファイル ノ ツイカ ト ヘンシュウ, JEB, 1/79
       WRITE ! DO ^MLOOK IF 'S KILL A, QRY, TEXT
       QUIT IF S>0 FOR I=0:1:ESIZ-1 SET A(I)=^(S+
         I) ;ヘンシュウ ノ トキ ハ エントリー ヲ トリダス
       ELSE SET A(0)=NAM FOR I=1:1:ESIZ-1 SET A
       (I)="" SET I=$PIECE(A(0), ",", 3), A(0)=$PIECE
         (A(0), ",", 2)_" "_ $PIECE(A(0), ",", 1)
       IF I'=" " SET A(0)=A(0)_","_I ;ショウゴウ ノ ツイ
       カ SET TEXT="NAME`ADDR1^ADDR2`ADDR3^CITY
         ^STATE^ZIP^TELE^FLAGS" ;ウナガシ ヨウ モジレツ
EDIT   FOR I=0:1 SET QRY=$PIECE(TEXT, "^", I+1)
         QUIT: QRY="" DO ELINE
REVIEW WRITE !!, A(0), ! ; ナマエ ノ レビュー
       FOR I=1:1:3 WRITE:A(I)'="" A(I), ! ; ジュウショ
       WRITE A(4), ",", A(5), "   ", A(6) , ! ; シ・シュウ・
         ユウビンバンゴウ
       WRITE:A(7)'="" A(7), "   " WRITE "[", A(8), "]",
         ! ; デンワ〔フラグ〕
VERIFY READ !, "F(ILE), E(DIT), OR I(GNORE)?    ", X
       GOTO FILE:X="F", EDIT:X="E", MEDIT:X="I"
         WRITE " ?" GOTO VERIFY
       ;
FILE   IF S<0 SET S=-S, ^(0)=^(0)+ESIZ FOR I=^(0):
         -1:S+ESIZ SET ^(I)=^(I-ESIZ)
       ;
       ; ココデ ナイブ ショシキ ノ ナマエ
```

```
        SET  NAM=$PIECE (A (0), " , ", 1)  FOR   I=1:1 QUIT:
          NAM'[" "     SET  NAM=$PIECE (NAM, "  ", 2, 99)
        SET A (0)=NAM_", "_$PIECE (A (0), "   ", 1, I-1)_", "
          _$PIECE (A (0), " , ", 2)
        FOR  I=0:1:ESIZ-1 SET  '(S+I)=A (I)  ;ファイル ニ
          データ ヲ イレル
        GOTO  MEDIT
        ;
        ; トクテイ ノ ギョウ ノ ヘンシュウ
ELINE   WRITE  !,QRY, "  "  WRITE:A (I)'=""  "<", A (I)
        READ  ">  ", X QUIT:X=""
        IF $LENGTH(X) LEN WRITE "[ナガスギ ケタオチ]"
          SET  X=$EXTRACT(X, 1, LEN-1)
        GOTO  UP:X="^", QUES:X="?"  SET:X="-"  X=""
          SET  A(I)=X QUIT
        ;
UP      SET  I=I-2  SET: I<0  I=99  QUIT
QUES    FOR  J=0:1:3  WRITE  $PIECE($TEXT(QT+J),";",2),!
        QUIT
        ;
QT      ; イレルモノ: アタラシイ データ, マタハ
        ;             フルイ データ ヲ タモツ ニハ CR,
        ;             ゲンザイ ノ データ ヲ ショウキョ スル ニハ
                      "-",
        ;             イゼン ノ ケンサク ヲ スル ニハ "^"
        ; ショウキョ プログラム ノ イリクチ
DEL     DO ^MLOOK  QUIT:'S
        GOTO  DEL:S<0  ; コノバアイ ナニモ シナイ
```

```
        FOR  I=S+ESIZ:1: ^(0) SET  ^(I-ESIZ)=^(I)  ;データ
          ヲショウキョ スル
        SET  ^(0)=^(0)-ESIZ  GOTO DEL
```

MLOOK ROUTINE (データ ケンサク ルーチン)

```
MLOOK   ; ファイル データ ノ ケンサク, JEB 1/79
        ; データ ヘノ ポインター S ヲ カエス
        ;                      <0 ハ データ ガ ミツカラナイ
                                 -S ハ データ ノ バショ
        ;                      =0 ハ CR ノ イレラレタ トキ
        ;               NAM ハ データ ノ ナイブショシキ ニ
                           ヨル ナマエ
        ; リャクシキ グローバル サンショウ
READ    SET  S=0  READ  !, " ナマエ ヲ クダサイ>", N  QUIT;
          N="" ; ナイブ ショシキ ノ ナマエ ヲ トル
        IF  N="?"  DO QUES  GOTO  READ
        ; セイ ヲ サガス (" " ノ イチ カラ モトメル)
        SET  NAM=$PIECE (N, ",", 1)  FOR  I=1:1  QUIT:
          NAM'[" "     SET  NAM=$PIECE(NAM, " ", 2, 99)
        SET  NAM=_","_$PIECE (N," ",1,I-1)_","_$PIECE
          (N, ",", 2)  ; ナイブ ショシキ
        SET  HASH=$ASCII (NAM) *100+$ASCII (NAM, 2) *
          10+$ASCII (NAM, 3)
        IF  '$DATA (^MAIL (HASH, 0))  SET  ^MAIL (HASH, 0)
          =0, S=-1  QUIT
        SET  N=^(0), LAST=$PIECE (NAM, ",", 1), I=0
        ;
        ; ナマエ ヲ サガシ チカイ データ デ ハイレツ IDX(I) ヲ
```

```
        ツクル
       FOR  S=1:ESIZ:N  GOTO:^(I)=NAM  FOUND  IF
        $PIECE(`(S), ",", 1)=LAST  SET  I=I+1, IDX(I)=S
       ;
       ; サガセナイトキ ハ ラシイモノ ヲ カク
       GOTO  NEW:'I  ; チカイ データ ガ ナイトキ
       FOR  J=1:1:I  WRITE  !, J, ?5, ^(IDX(J)), ?45,
        ^(IDX(J)+1)
WHICH  READ  !, "データ バンゴウ ヲ イレル カ N(コレラデナイ)ト
        イレル> ", X  IF X="N"  GOTO  NEW
       IF  X'?1N. N!(X>I)  WRITE  "?"  GOTO  WHICH
       WRITE  !  SET  S=IDX(X)  GOTO  QUIT
       ;
FOUND  WRITE  "     ",^(S+1)  ; ADDR1 ヲ カキダス
       READ  "      OK?    ", X  GOTO  QUIT:X?1"Y". E
        HERE ; YES デ ナケレバ ツイカ ヲ オコナウ
       ;
NEW    WRITE  "ミアタラナイ  ", !
       IF  N=0  SET  S=1  GOTO  HERE
       FOR  S=1:ESIZ:N  GOTO: ^(S)]NAM  HERE  ;
        タダシイバショ ヲ ミツケル
       SET  S=S+ESIZ  ; サイゴ ニ ツケル
HERE   SET  S=-S  ; アタラシイ データ トシテ マーク スル
QUIT   KILL  N, IDX, LAST QUIT
QUES   WRITE  !, "ナマエハ "メイ" "セイ" "ナカ" "ショウゴウ"ノ
        ジュン デ イレナサイ ", !  QUIT
```

MPRT ROUTINE (ファイル カキダシ ト ラベル ノ ルーチン)

```
MPRT  ; シュツリョク ルーチン, JEB 1/79
      ; ラン ノ カズ ト ギョウ ノ カズ ガ ヨウキュウサレル
      ; PRINT オヨビ LIST ガ イリクチトナル
PRNT  SET BOOL=1 GOTO OUT ; スベテ ヲ エラビ シュツ
        リョク スル
LIST  SET BOOL=1 READ !, "ジョウケン <1> ヲ エラ
        ビナサイ", X IF X'="" DO MAKB
UFLG  SET UF="" READ !, "フラグ ヲ コウシンシナサイ",
      UF IF UF' ?. UN WRITE "オオモジ アルファベット カ
        スウジデ イレナサイ" GOTO UFLG
OUT   READ !, "カキダシ ソウチ ハ ? ", IO IF IO="" SET
        IO=$I
      WRITE ! SET HASH=-1, (S, N)=0 USE IO WRI-
        TE # DO NEXT
LOOP  IF S<0 WRITE !# USE 0 GOTO KILL
      FOR L=1:1:NL FOR C=1:1:COLS SET BUF (C, L)
        ="" ; バッファー ヲ クリヤ スル
      FOR C=1:1:COLS DO BUFFER, NEXT QUIT:S<0
      FOR L=1:1:NL WRITE ! FOR C=1:1:COLS WRITE
        ?C-1*40, BUF (C, L)
      WRITE !! GOTO LOOP
      ;
KILL  KILL BOOL, UF, S, N, L, C, F, HASH, NAM, BUF, NOT,
        IO QUIT
      ; ツギノ ファイル イリクチ ヲ ミツケル. Sガ ソレヲ サシシ
        メス (ナケレバ <0)
```

```
NEXT   SET S=S+9 GOTO NEW:S>N SET F=^(S+8) ;
         フラグ ノ ワード ヲ モトメル
       IF @BOOL QUIT ; カンセツシテイ デ アル
       GOTO NEXT ; ツギ ヲ ココロミル
NEW    SET S=-8, HASH=$NEXT (^MAIL (HASH)) QUIT:
         HASH<0 ; モウ データ ハ ナイ
       SET N=^(HASH, 0) GOTO NEXT ;コノ バケット ヲ
         ミテ ミル
       ;
       ; ヒトツ ノ データ ヲ バッファー ショシキ ニ スル
       ; I ハ シュツリョク ノ サイノ ギョウ バンゴウ
BUFFER SET I=1, NAM=^(S), BUF(C,I)=$PIECE(NAM, ",",
         2)_" "_$PIECE(NAM, ",", 1)
       IF $PIECE(NAM, ",", 3)'="" SET BUF(C, I)=BUF
         (C, I)_","_$PIECE(NAM, ",",3) ; ショウゴウ ガ ア
         レバ トリアツカウ
       FOR J=1:1:3 IF ^(S+J)'="" SET I=I+1, BUF(C,
         I)=^(S+J) ; カラモジ ハ ムシ スル
       SET I=I+1, BUF(C, I)=^(S+4)_","_^(S+5)_"  "_
         ^(S+6) ; シ・シュウ・ユウビンバンゴウ
       IF FN="P" SET BUF(C, I+1)="" SET :^(S+7)'
         =""BUF(C, I, 2)=^(S+7)_" "
       IF SET BUF(C, I+2)=^(S+7)_"["_F_"]"
       ELSE SET^(S+8)=F_UF ; アタラシイ フラグ ヲ セッ
         トスル
       QUIT
       ;
       ; X カラ BOOL ジョウケン ヲ ツクル
```

```
MAKB SET BOOL="",NOT="" FOR I=1:1:$LENGTH(X)
       SET C=$EXTRACT(X, I) DO CHAR
     QUIT
     ;
CHAR IF C="'" SET NOT="'" QUIT
     IF C?1UN SET BOOL=BOOL_"(F"_NOT_"[" " "
       _C_" " ")", NOT="" QUIT
     SET BOOL=BOOL_C ; エンザンシ デ アルコト
     QUIT
```

MFLG ROUTINE (フラグ ヘンシュウ ルーチン)

```
MFLG ; フラグ ヘンシュウ ト ソノ ディスプレイ, JEB, 1/79
READ READ !, "フラグ>", FLG ; フラグ メイ ヲ イレル
     GOTO LIST:FLG="*" QUIT:FLG=""
     IF FLG'?1UN WRITE "アルファベット カ スウジ ノ 1
       モジ デ!" GOTO READ
     SET FA=$ASCII(FLG), X="" ; X ガ ゲンザイ ノ アタ
       イトナル
     IF $DATA(^MFLG(FA)) SET X=^(FA) WRITE !,
       SET "  ", X, "  ", !
     READ !, " > ", Y, ! IF Y'="" SET:Y="-" Y=""
       SET ^MFLG(FA)=Y
     GOTO READ
     ;
     ; フラグ ヲ リスト アップ スル
LIST READ !, " シュツ リョク ソウチ?", IO IF IO="" SET
       IO=$I
     USE IO WRITE #!, ?30, "フラグ ノ リストアップ", !! ;
```

```
        ココデハ 72ラン ノ タンマツ ヲ ソウテイシテイル
      SET FA=-1, X=$DATA (^MFLG (0))  ; リャクシキ
        グローバル ヲ セッテイ スル
      FOR I=0:0 SET FA=$NEXT(^(FA)) QUIT:FA<0
        WRITE $CHAR(FA), ?5, ^(FA), !
      WRITE # USE 0 GOTO READ
QUIT  WRITE ! KILL FLG, FA, X, Y, IO QUIT
```

以上に掲げたプログラムのリスティングは多くの経験をつんだ MUMPS プログラマによって用いられる手技を利用している．それらをここで解読することができれば，そしてそれらを自身のコンピュータに書き込みながら部分部分を実行して分析することは有益なことであろう． MLOOK は新しいデータをファイルに書き込むときのユーティリティであるので，最初に MLOOK を入れて，その後に MEDIT を入れるとこれが MLOOK を用いていることがわかる．これらの練習を体験するときに，本書のすべての章を復習すれば，これらのルーチンに用いられている機能をよりよく理解できるであろう．

郵送リスト パッケージの出力例

以後の頁に郵送リスト パッケージとの対話の実例をみせる．以上のすべてのルーチンを自身のコンピュータに装備してみれば，以下にみる対話と同じことを実演してみることができよう．

（ユーザーの入力には下線を施す）

```
D ^MOPT
ユウソウ リスト パッケージ
サギョウハ ?> ?
              ADD/EDIT ENTRY
              DELETE ENTRY
              FLAG EDIT
```

```
            PRINT FILE
            MAILING LIST
サギョウハ ? >  ADD/EDIT ENTRY

NAME > ?
ナマエ ハ "メイ" "セイ" "ナカ" "ショウゴウ" ノ ジュン デ イレナサイ
NAME > JOHN SMITH
NAME <JOHN SMITH>
ADDR 1 > HAL INC.
ADDR 2 > BOSTON
ADDR 3 >
CITY > MASS
STATE > MA
ZIP > 02217
TELE > 617-555-9876
FLAGS > 1

JOHN SMITH
HAL INC.
BOSTON
MASS, MA 02217
617-555-9876 [1]

F (ILE), E (DIT), OR I (GNORE) ? E
NAME <JOHN SMITH>
ADDR 1 <HAL INC. >
ADDR 2 <BOSTON> MAIN STREET
ADDR 3 >
CITY <MASS> BOSTON
STATE <MA>
```

```
ZIP <02217>
TELE <617-555-9876>
FLAGS <1>

JOHN SMITH
HAL INC.
MAIN STREET
BOSTON, MA 02217
617 555-9876
F (TLE), E(DIT), I (GNORE) ? F
NAME > JOHN SMITH
1    SMITH, JOHN                                HAL INC.
データバンゴウ ヲ イレルカ N(コレラデナイ) トイレル> N ミアタ
  ラナイ
NAME  <JOHN SMITH>
ADDR 1  > SYSTEMS CONSULTING
ADDR 2  > ROOM 142
ADDR 3  > BROADWAY
CITY  > NEW YORK
STATE  > NY
ZIP  >
TELE  >
FLAGS  >

JOHN SMITH
SYSTEMS CONSULTING
ROOM 142
BROADWAY
NEW YORK, NY
```

[ ]

F (ILE), E (DIT), OR I (GNORE) ? F

NAME >

サギョウ ハ? > PRINT FILE （注：すでに他の名前が入力済みと仮定する）

カキダシ ソウチ ハ ?>

JACK BOWIE, SC. D
MAIL STOP #641
MITRE CORP.
1820 DOLLEY MADISON BLVD.
MCLEAN, VA 22102
703-827-6667 [A1]

JOHN JONES, PH. D
1234 STARS END
NEW YORK, NEWYORK 12345
212-555-1212 [ ]

JOHN SMITH
HAL INC.
MAIN STREET
BOSTON MA 02217
617-555-9876 [1]

JOHN SMITH
SYSTEMS CONSULTING
ROOM 142
BROADWAY
NEW YORK, NY
[ ]

RICHARD WALTERS, PH. D
DEPT OF COMMUNITY HEALTH
SCHOOL OF MEDICINE
UNIVERSITY OF CALIFORNIA
DAVIS, CA 95616
916-752-2793 [A]

サギョウ ハ ? > FLAG EDIT

フラグ > 2

>RECEIVED SECOND MAILING

```
フラグ > *
シュツリョク ソウチ？>
                    フラグ ノ リストアップ
1    RECEIVED FIRST MAILING
2    RECEIVED SECOND MAILING
A    AUTHOR
フラグ >
サギョウ ハ？ > MAILING LIST
<1> トスル ジョウケン ヲ エラビナサイ  > 1
フラグ ヲ コウシン シナサイ  > 2
カキダシ ソウチ ハ？ >
JACK BOWIE, SC.D                JOHN JONES, PH.D
MAIL  STOP  ♯641                1234 STARS END
MITRE CORP.                     NEW YORK, NEWYORK 12345
1820 DOLLEY MADISON BLVD.
MCLEAN VA 22102

JOHN SMITH                      RICHARD WALTERS, PH.D
SYSTEMS CONSULTING              DEPT OF COMMUNITY HEA-
ROOM 142                        LTH
BROADWAY                        SCHOOL OF MEDICINE
NEW YORK, NY                    UNIVERSITY OF CALIFORNIA
                                DAVIS, CA 95616
サギョウ ハ？ > ADD/EDIT ENTRY
NAME > SMITH, J
1    SMITH, JOHN,               HAL INC.
2    SMITH, JOHN,               SYSTEMS CONSULTING
データ バンゴウ ヲ イレルカ N(コレラデナイ) トイレル >2
```

```
NAME  <JOHN SMITH>
ADDR 1  <SYSTEMS CONSULTING>
ADDR 2  <ROOM 142>
ADDR 3  <BROADWAY>
CITY  <NEW YORK>
STATE  <NY>
ZIP  >
TELE  >  212-555-1212
FLAGS  <2>

JOHN SMITH
SYSTEMS CONSULTING
ROOM 142
BROADWAY
NEW YORK, NY
212-555-1357  [2]

F (ILE), E (DIT), OR I (GNORE) ? F
NAME >_
サギョウ ハ ? > PRINT FILE
カキダシ ソウチ ハ ? >_
JACK BOWIE, SC.D                  JOHN JONES, PH.D
MAIL STOP ♯641                    1234 STARS END
MITRE CORP                        NEW YORK, NEWYORK 12345
1820 DOLLEY MADISON BLVD.         212-555-1212  [2]
MCLEAN, VA 22102
703-827-6667  [A12]

JOHN SMITH                        JOHN SMITH
```

HAL INC.
MAIN STREET
BOSTON, MA 02217
617-555-9876 [1]

SYSTEMS CONSULTING
ROOM 142
BROADWAY
NEW YORK, NY
212-555-1357 [2]

RICHARD WALTERS, PH.D
DEPT OF COMMUNITY HEALTH
UNIVERSITY OF CALIFORNIA
DAVIS, CA 95616
916-752-2793 [A2]

要　約

前頁までの出力をよくしらべてみる際に第 3 頁目のメイル リストでは BOSTON の SMITH 氏は出力に含まれていないことに気がつくであろう．これは，彼のフラグの中に [1] が入っており，出力からこれらのついた人々を除外するように指定がなされているからである．またこのときフラグが更新されて，彼以外の人はこのリストに書き出されたときの副産物として 2 というフラグが付け加わっていることに注意すべきである．

この入門書の役割は読書が MUMPS 言語に論理的で順序立った紹介をされることを助ける点にある．しかしながらどんなプログラミング言語でも実践以上によい入門の方法はないのであり，一度本書を読み終えたあとでも，新しい挑戦につきあたるたびにこの入門書をひもとくことがあるはずである．そのようなときには，本書巻末にある言語要素索引簿をひいて，望む言語機能をさがすとよい．

最後の 2 章すなわち第 18 章と第 19 章は折にふれて復習し，そのつど MUMPS プログラミングの技法をよりよく理解し，これらの例を相当改良することができるまでになるであろう．すなわち，これらの例は本来入門者のために書かれたもので，プログラム技法として最適のものが選ばれたわけではないか

らである．

本書は，読者が自身の用いる MUMPS 機種を扱う場合に特にその MUMPS 装備のもつ非標準要素（装備依存機能）をも同時に習熟できるよう，装備特異的言語要素について読者にそのつど学習をうながしてきた．本書の中に変化しつつある各種の装備の装置依存要素 (implementation dependent fatures) を表にまとめることは困難であるので，各ベンダーが本入門書に読書にとって必要な特定装備の拡張～装備特異要素の1章を挿入することが強く望まれる．読者は自分の用いる MUMPS 装備の拡張～装備特異要素を次の第19章の終りの頁に抜粋しておくのも一法であろう．最後に，言語は人にそれを教えるときほど自分の知識の骨格を固めるよい機会はなく，本書の著者らも経験上このことを実証することができたことを付記しておく．

謝　辞

この章で用いた例は Massachusetts General Hospital のコンピュータ科学研究所（LCS）の Robert Rees の作成による同名パッケージに負うところが多い．著者らは彼の多くの有意義な，特にフラグ操作の領域での示唆に対し感謝の意を表する．

パタン暗記 p98

第 19 章

標準 MUMPS ガイド

はじめに

この章は標準 MUMPS の簡潔なまとめと，わかりやすい言語機能の例示を本入門書の読者に提供しようとするものである．本章は標準 MUMPS に関するすべての命令語，関数，演算子，および他の機能について要約し，MUMPS 開発委員会（MDC）が定義し，米国標準局 (National Bureau of Standard) が出版した Handbook 118 にもとづいている．しかし，本章でもって MDC/ANSI の標準 MUMPS 仕様書にかえて理解することには十分な注意がなされるべきであり，標準 MUMPS の詳細に関しては MDC刊 J. T. O'Neill 編 MUMPS Language Standard, ANSI X11. 1-1977 (1977) または，「標準マンプス言語マニュアル」(MDC 刊, MUMPS Programmers' Reference Manual, 若井，田中，嶋訳，コロナ社発行，1977) を参照されたい．上級プログラミング技術についての解説を得たい場合は，MUMPS ユーザーズ グループ 刊, Beckley, R. E., and Bridger, D. A., Advanced MUMPS Techniques, 1977 がすすめられる．

MUG ないし MDC の文書資料に関してさらに詳細を知りたい場合には，名古屋市東区布池町 32, MUG ビル内日本マンプスユーザーズグループ事務局宛請求するとよい．

本章を通じて説明の一部に角括弧〔 〕の施されているところがある．これによって標準仕様の定める制限ではないが，MUMPS プログラムの他機種への移送性を念頭におく場合従わなければならない制限を示している．この，〔MUMPS 移送基準〕は互換性を目標とした場合守られるべき基準となる．

この標準 MUMPS ガイドは Joan Zimmerman が National Center for

Health Services Research, DHEW との HS-01540 契約番号によって完成した「標準 MUMPS ポケットガイド」(若井，嶋訳，日本マンプス ユーザーズ グループ事務局発行) を，Joel Achtenberg が拡張するために手を加えたものにもとづいた．この拡張に際して Dan Schullman の広範なコメントと Joan Zimmerman からの助言があったことを付記し，同時に MUMPS 共同体からの多くの要求と示唆，特に Robert Greenfield, David Shereretz, Robert Stimac, Anthony Wasserman らの助力によるところの多かったことを併記し感謝する．

この章は，MUMPS ユーザーズ グループ 1978 年 9 月版権所有のものであり，引用に際しては MUG からの書面による許可が必要である．

データの型および値

まず第一に考慮しておかねばならないものに標準 MUMPS で扱うデータの型がある．これは非常に簡単であり，データの型はただ一つ，可変長文字列であり，文字として印字できる ASCII 文字（JIS 文字）である〔移送基準を満足させる現在の文字列長は 255 文字までと制限されている〕．

数値は単に文字列の特別な場合としてみなされる．そして MUMPS はどんな文字列をも数値として解釈する厳密な規則をもっている．数値（すなわち数としての値）は小数を含んでもかまわない．しかしある種の算術演算子が数値の特別の場合として整数値を出すことがある．したがって MUMPS にはいかなる数（また敷衍して，すべての文字列）をも整数として解釈する規則をもっている．関係演算子と論理演算子（下記）は真偽値を出す．真偽値には 2 個あり，整数 0 は “偽” を整数値 1 は “真” をあらわす．真偽解釈の規則は下記の演算子で解説する．

〔移送基準を満足させる数値は有効桁数 9 桁をこえてはならない．絶対値は 10^{25} と 10^{-25} の間ないしゼロである必要がある〕

変　数

変数とはその値が変化することのできる1つの実体である．変数の型には3種あり，ローカル変数はユーザー個人にとって特異的なものでそのユーザー個人がそれを検査しまたはかえることのできるものであり，グローバル変数は許されたものなら複数のユーザーが値を検査しまたはかえることができる．第3の変数である特殊変数は MUMPS システムがその値をかえユーザーが直接値をかえることはできない．

おのおのの変数は変数名によって参照される．ローカル変数名とグローバル変数名は第1文字が“%”またはアルファベット文字で，それ以下の文字がどのアルファベット文字ないしは 10 個の数字のどれであってもよい．それらの名前はどんな長さであってもよい（移送基準を満足させるには名前ははじめの8文字で識別でき，しかも，小文字アルファベットを含まないものでなければならない）．グローバル変数の名前は頭に上向き矢印（^ または↑）記号をもちたとえば ^MUG という形であらわされる．特殊変数の名前は $TEST というように，その頭にドル記号（$）をもつ．特殊変数についてはあとの節で解説する．

ローカル変数とグローバル変数はそれらに添字を付けることによって一連の値を配列という集合にまとめることができる．たとえば，GAME, SET, MATCH という変数は，それぞれ A(1)，A(2)，A(3) というふうに置きかえてみると使いやすくなる場合がある．この例では添字は1つのレベルしか使っていないが，必要ならばもっと多くの添字を使うことができる．たとえば A(1, 4, 13) とすれば3つのレベルの添字を用いたことになる．これらの変数で A(1)，A(1, 4, 13) はAの下位節であり，また A(1, 4, 13) は A(1) の下位節であるという．

変数は添字をいくつもってもよい．添字の取る値は許された負でない整数（すなわち，0 から 999999999 までの）のどれをとってもよい．

略式グローバル参照は，変数名を省略し，しかも添字のいくつかを省略して

1個のグローバル変数を表現する速記的構文規則である．略式グローバル参照では並記される添字のうち，第1番目の添字はもっとも直前に参照されたグローバル参照で用いられた添字の最後の添字レベルに暗黙に一致するものであることが決まっている．それゆえ，はじめに ^X(1) というグローバル参照があり，次いで ^(2,3) という略式参照があれば，それは ^X(2,3) の値を参照することになる．"直前のグローバル参照"とはいかなるグローバルへのどのような参照法であってもよいことに注意．略式グローバル参照はグローバル変数にポインターがおかれているのを利用した速記術であるが，$NEXT 関数を用いる際には，略式参照用ポインターに，思いがけない値を残遺させることがあるので特別困難を生ずることがある（なお詳細は後述の $NEXT を参照のこと）．

定字列

定字列 (literal) は変化しない値をもち直接解釈の対象となる．定字列には数字列 (numeric literal) と定文字列 (string literal) がある．数字列は1個の仮数部があり，それに任意に文字"E"に導かれた指導部がつく（例：0.34, −1456, 10E5, 1.2E−7)．定文字列は引用符ではさまれた一連の文字である．空文字列という値を示すには引用符を2個連続させる（""）．定文字列の中に引用符にはさまれた引用形を用いる場合は，内側の引用符を二重にする ("THE ""KNOWN"" VALUE")．この場合2個の引用符は文字列長を決める際2文字でなく1文字として数えられる．定文字列は長さを0文字を含む ASCII (JIS) の印字できる 95 個（159個）の文字と符号からなり引用符ではさまれているが，文字列 (string) という場合には ASCII (JIS) の 128 文字 (224 文字) を含みうるものである．

演算子

標準 MUMPS の演算子は7種の型に分類される．

算術一項演算子 (arithmetic unary operators)

算術二項演算子（arithmetic binary operators）

算術関係演算子（arithmetic relational operators）

文字列二項演算子（string binary operators）

文字列関係演算子（string relational operators）

論理演算子（logical operators）

間接指定（indirection, @）

これらの演算子はそれらの種類によって，とられる引数の解釈が異なる．3種の算術演算子は，それぞれのかかる引数を演算をおこなう前に数値解釈をする．これは算術演算子のかかる文字列のもっとも左端が指数表示であれ（例：86E5)，小数表示であれ（例：－182.45）または整数表示であれ（例：1964）その部分を数値として取ってくることを含む．もしこのような形式でその文字列引数がはじまらない場合は数値解釈はゼロとなる．値を整数解釈する場合にはまず数値解釈をしたのち，小数部分を切りおとすことでおこなわれる．論理演算子は最初にそれのかかる引数を最初に数値解釈し，その結果が0ならば解釈は“偽”となり，それ以外の場合に“真”と解釈する．2種の文字列演算子は標準 MUMPS ではすべてのデータは文字列であるゆえに，それらのかかる引数に特に解釈を加える必要はない．

文字列	解釈演算 数値	整数値	真偽値
“810”	810	810	1
“98 ポンド”	98	98	1
“”（空文字列）	0	0	0
“　35”	0	0	0
“86+9”	86	86	1
“PAGE　10”	0	0	0
“−8.4”	−8.4	−8	1
“86E−1”	8.6	8	1
“−−−9”	−9	−9	1

"−0" 0 0 0

算術演算子，文字列演算子，論理演算子のおのおのについては例とともに別表にして示してある（演算子一覧表（表 19·1））．間接指定は型式は演算子であるが，その用い方は特異的であり，以下に項を改めて解説する．

間接指定

間接指定ではデータの値が MUMPS コードとして解釈される．間接指定は文字 "@" に続く1個の式であらわされる．間接指定が生起するたびに式の値がそこに代入されその行の残りが解釈される前に MUMPS コードとして扱われる．この代入は一時的であり，間接指定の生起するごとにおこなわれる．これは，MUMPS コードの同一部分が何度も実行されるごとに式の異なった値を代入されて結果をかえてゆくことを許す．さらに，間接指定は入れ子（ネスティング）をつくることもできるのでなおも柔軟性をもたせることができる．基本的に間接指定には3つの型がある：

引数間接指定では命令語の引数の場所に生じ，命令語の1つまたはそれ以上の引数の値に対して間接的な指定をする．たとえば，

例：S X(1)="STATE(I)", X(2)="CAPITAL(I)", X(3)
="POPUL (I)"
F I=1:1:50 F J=1:1:3 W $P(X(J), "(", 1), ":" R
@X (J), !

は 50 の州の名前，主都，および人口の項を指定し，その値を入力させる．

例：S PRINTER=3, TERMINAL=0
R !, "DEVICE TYPE ? ", DEV OPEN @DEV

この例ではユーザーは PRINTER または TERMINAL という文字列を入力すると，適当な端末が使えることになる．

例：S USERCODE="B"
S @("AA"_USERCODE_"=99")

名前間接指定では変数名が発生しうるいかなる場所にも生起し，間接指定の

とる値は1個の完全な変数名であり，それは添字を含めてとりうる．

例：S　AGE=40, RACE="コーカシヤ", SEX="オンナ"

R　!, "ドノ ヘンスウ ノ ナイヨウ ヲ ミタイデスカ? ", X

W　!, X, "ノ ナイヨウ ハ ", @X, " デス."

ではユーザーが SEX という文字列を入れれば SEX ノ ナイヨウ ハ オンナ デス．という出力をみるはずである．

パターン間接指定では間接指定がパターンの場所で生起し，その値は1個のパターンである．

例：S　X(1)="1" "$" "1N.N", Y(1)="MONEY"

S　X(2)="5U.U", Y(2)="WORD"

FOR　I=1,2　IF　STRING? @X(I)　WRITE　!, Y(I) Q

では，もし STRING が $ 符号を頭においた1桁ないしそれ以上の数であれば MONEY と出力し，STRING が5個かそれ以上の大文字であれば WORD という結果を出す．XECUTE 命令もまた間接指定の実行をおこなうことに注意（後述）．

式

MUMPS でもっとも簡単な式は変数，定文字列，定数字列，または関数（関数については後述する）である．これら4つの型の式をそれぞれ例示すると：

変数

"文字列"

45.73

$LENGTH (XYZ)

のごときものであり，このような簡単な式を単元式（式子）という．複雑な式は複数個の単元式（式子）を算術演算子ないし，その他の演算子によって連結してつくることができる．たとえば，

SUM/TOT

SEX= "オンナ"

"セイネン"_"ジダイ"

すべての MUMPS 式は左から右に向けて評価される．演算子には優先順位があり，まず一項演算子が間接指定の以前に，そして間接指定は二項および関係演算子よりも前に実行される．しかし二項および関係演算子の間では優先順位はない．括弧（ ）を用いれば評価の順位をかえることができる．

表 19・1 演算子一覧表

演算子記号	意　味	例	結果およびコメント
1. 算術一項演算子			
+	数値解釈をおこなう	+2	2
		+"34A"	34
−	数値解釈をおこない負の値とする	−A	−7（もしAの値が7となれば）
		−"34A"	−34
2. 算術二項演算子			
+	和を出す	2+7	9
		A+3	Aの値プラス3
−	差を出す	2−7	−5
		A−B	AとBとの差
*	乗を出す	2*7	14
／	商を出す	7／4	1.75
＼	商の整数部分以下を切り捨てる	7＼4	1
		A＼5	0（Aが −5 より大で5より小のとき）
#	右引数に関する左引数の法（モジュロすなわち余り）を出す	11 # 3	2
		11 # −3	−1
		−11 # 3	1
		−11 # −3	−2
3. 算術関係演算子			
<	小なり	MINIMUM<8	もしMINIMUMが8より小さければ 1，そうでなければ 0
>	大なり	MAXIMUM>LIMIT	もし MAXIMUN の値がLIMIT より大きければ1，そうでなければ0
4. 文字列二項演算子			
_	前後の文字列の結合	"GO"_END	"GOING"（もし END="ING"） "GOES"（もし END="ES"） "GO"（もし END=" " ならば）

演算子記号	意　　味	例	結果およびコメント
5. 文字列関係演算子			
=	ひとしい (等号)	NAME= "JONES"	NAME の値が "JONES" ならば 1, そうでなければ 0
[	含(含む, コンテインズ)	NAME ["SON"	NAME の値に "SON" が含まれれば 1, そうでなければ 0
]	追従(続く, フォローズ)	NAME] "M"	NAME の値がASCII文字表で"M"よりあとにくるなら 1, そうでないと 0
?	パターン照合 (パターンのマッチ)	NAME ? 2A	NAME の値が正確に 2 個のアルファベット文字なら 1, そうでないと 0
		NAME ? 1A. A	NAME の値が 1 個のアルファベット文字なら 1, そうでないと 0

パターン照合演算子で許されたパターンコードは次のごとし

A (26 の大文字, 26 の小文字アルファベット)

C (33 の制御文字)

E (すべての 128 ASCII 文字)

L (26 の小文字アルファベット)

N (10 個の数字)

P (33 の句読文字)

U (26 の大文字アルファベット)

"定文字列" (引用符に入った文字列)

(日本の機種はカナ文字を装備しているが, 上記コードにない文字Kでカナをあらわす傾向にあり, 日本 MUG ではその他にカナ大文字, カナ小文字, 特殊文字, 句読文字などのコードについてMDC にそれらの日本用確保のための意見を求めているが, MDC は将来の MUMPS と, 多種国語を考慮して, 日本ではたとえばカで全カナ, オでカナ大文字, コでカナ小文字, クで句読文字, トで特殊文字 (ー, ˝, ˚) をあらわすコードとしてはどうかと提案が出されている. 日本語はひらがななどの問題もあり, 標準的な日本語用 MUMPS のパターンコードを一本に統一する動きがあり, 大櫛陽一委員長のもとにまとめられつつある (以上訳注).)

各パターン型の生起する回数は正確な整数をその前において指定することができる. 何回生起してもよいときは, その前に " . " をおく. ゆえに "1N" は正確に 1 回数字が生起することを, ". P" は何回句読文字が生起してもよいし, それが 0 回でもよいことを示す. 許されるパターンがグループ化してもよい. たとえば ". A" はアルファベット文字を, ". P" は句読文字をチェックするが, ". AP" はアルファベット文字または句読文字をさがす. ". PA" と書いても効果は同じ. しかし定文字列はパターンコードと結合してはならない (例: . A"7" は許されない).

演算子記号	意　　味	例	結果およびコメント
6. 論理演算子			
&	アンド (論理積, かつ)	A & B	AもBも真のときに1, AとBのどちらかが偽ならば0
!	オア (論理和, または)	C ! D	CとDのどちらかが真ならば1, CもDもどちらも真でなければ0

演算子記号	意　　　味	例	結果およびコメント
′	ノット（否定，ではない） 否定（′）はどの算術演算子や文字列関係演算子と結合して用いてもよい（たとえば，A′=3 は ′(A=3) と等価である	′E	Eが偽ならば1 Eが真ならば0
7. 間接指定符号 @	間接指定を使えばデータの値を MUMPS コードとして実行することができる 間接指定は文字@で示され，その後に単元式（式子）が続く．間接指定のある行の残りの部分が解釈される前にその間接指定のところにその式の値が代入される．間接指定には次の3つの基本型がある 1. 引数の間接指定．これは間接指定が命令引数の場所に置かれる場合で，この場合式の値は1つまたはそれ以上の引数として完全な形になっていなくてはならない． 2. 名前の間接指定．間接指定は変数名を置くことのできる場所ならばどこにでも置くことができるが，この場合間接指定の値は1つの変数名として完全なものでなくてはならない 3. パターン間接指定．パターンのかわりに間接指定を置いてもよいが，この場合その値はパターンになっていなくてはならない （完全な形の MUMPS 命令行の間接指定としては XECUTE 命令を用いうることに注意）	@X	X の値が MUMPS コードとして実行される

命令語

1個の命令語は実行されるべき操作内容を定義するものである．命令語には普通（ただし常にではない）引数（またはコンマで区切られた一連の引数並記）が続き，それらに対して命令の操作が加えられる．ほとんどの命令語は省

略してその頭文字１字だけであらわすこともでき，また完全に綴ってもよい．一部分だけを省略することは許されない．このように“B”は“BREAK”の合法的な省略型であるが，“BR”は許されない．指定されていない頭文字（訳注：A，J，M，N，P，T，Y である）は将来利用するために確保されているものである．

MUMPS プログラムでは１つの行内にある１個の命令語とその引数（群）は，同一行でもし次に命令語がくる場合には，その間を１個のスペースで分ける．また命令語はそれの引数との間に１個のスペースをおいて用いる．このことは，引数なしの命令語では次の命令語との間に正確に２個のスペースを必要としていることを意味している．一般に何らかの命令語Ｚが引数ＡとＢをとる場合，Ｚ　A, Ｂという形はＺＡ　ＺＢと等価である．注釈文はセミコロン（;）を頭にして命令語を置きうる位置に挿入することができる．この場合その行のセミコロン以後は注釈文とみなされ実行はされない．ただし $TEXT で呼び出されるラインのセミコロンは，データそのものとなるので，どのように処理されるかは，プログラムによってことなる．

命令語の時間制限（time out）

時間制限は命令語 LOCK，OPEN および READ の引数とともに用いられ，それらの操作が完了するのを MUMPS が待っている最大時間を指定する．時間制限が用いられる際はこれらの命令語の引数の１つの直後にかかれ，コロンと整数値をとる式を用いる．

READ　ANSWER : TIME

では ANSWER が入力されるまで，TIME 秒までは待っている．コロンに従う数値が正の数ならば操作の終了までにその秒数を最大限として待つことを意味し，数値がゼロかまたは負の値のときは実行は待たずに続行される．もし，LOCK, OPEN または READ 命令が時間制限を付けており，その時間制限内に満足されたときには，特殊変数 $TEST（後述）が１と設定され，さもなければ $TEST は０と設定される．何の時間制限も付いていないときには，$T-

EST は影響をうけず，命令語に関する条件（たとえば，READ 命令の入力メッセージが完了する）が満足されてから命令語が実行される．したがって命令行の実行は無制限に棚上げされうる．

命令語と引数に対する後付け条件

IF 命令は，IF 命令に続く残りの命令行が実行されるか否かの条件を付け加えて用いることができる．またほとんどの命令語にもそれぞれに必要な条件がコロン（:）に続く真偽値式を命令語の直後に付けて用いることができる．例外となる命令語は ELSE と IF である．たとえば，

SET : A=3 NEXT=47 GOTO LINE

では，A が 3 という値をとるときにのみ 47 という値が NEXT に割り付けられることを意味する．しかしどんな場合でも GOTO 命令は常に実行される．この際 SET 命令が後付け条件化されているという．命令語は直後のコロンに続く式が真である場合にのみ実行される．IF 命令が，あとに続くすべての命令行の実行に影響するのに反して，この条件は後付け条件化された命令行のみが影響されることを銘記すべきであり，条件がどうであれ右側に続く命令文のすべてがふつうに実行されるのである．また後付け条件では $TEST 値は影響をうけない．

DO, GOTO および XECUTE 命令はそれらの引数の後付け条件法を許し，同時にまた命令語そのものの後付け条件法も許す．このことはこれら 3 つの命令語の引数のあとにはコロン（:）と真偽値式を付けてよいことになる．もし真偽値式が真であれば，その引数が採用され偽であればその引数に用いられない．たとえば，

DO FIND : A=100, NEXT

ではまず A が 100 という値の場合のみ FIND というラベルのついた行を実行しはじめ，そうでないときは必ず NEXT というラベルをもつ行から実行をはじめる．このように，後付け条件のついたその 1 個の引数のみが影響をうける．後付け条件の付いた引数以外のすべての引数と，その右にある命令語-引

数の組み合わせは，通常に実行される．

次の例は命令語も引数も後付け条件化された場合である：

DO : B="OK"　FIND : A=3, NEXT

表 19・2　命令語一覧表

命 令 語	例	説　明
BREAK	B	BREAK は直接モード（標準 MUMPS には含まれない）のプログラミングやデバッグが容易になるように MUMPS 内のアクセス点を与える．プログラムの実行はある特定の信号（これは標準 MUMPS で指定されていない）を受けるまで中断される．BREAK はふつう引数をもたないが，言語装備に特有な引数を用いることのできる場合もある
CLOSE	C 4 C MT1: (devparms)	CLOSE は引数として並記された装置の占有権を解除する．たとえば装置 4 または MTI の占有権を解除する．装置名のあとに，必要な場合は装備特有の装置パラメータを付加することもできる．現用装置が CLOSE されると特殊変数 $IO は空となるか欠損代用値をもつが，これも装備特有である
DO	D NEW, OLD D ^A D LINE^ROUTINE D @VARIABLE	引数として並記された行ラベル（たとえば，NEW, OLD）が順次さがし出されて，おのおののラベルに続くプログラム コードが "終結点"（後述）に達するまで実行される．^Aのようにラベルの頭に "^" がつく場合には，"^" に続くルーチン名（A）が実行される．"^" の前に行ラベル（たとえば，LINE）が置かれない限り，実行はルーチン名のついた第 1 行からはじめられる．後者の場合実行は ROUTINE という名のルーチンのラインというラベルの行からおこなわれる．そして QUIT が実行されるか，ルーチンの最後にくると DO の引数の次の引数かまたは次の命令行にもどる（ただし，このルーチンの最後がはじめの DO の行き先にあった次の DO によるルーチンの終りである場合や，同じく行き先にある FOR 命令の中での QUIT やルーチンの終りである場合は異なる)．間接指定（例，@VARIABLE）は変数に入っているラベルから，行ないしルーチンをはじめるために用いることができる
ELSE	E	ELSE は特殊変数 $TEST（後述）の真偽値に従って条件付きの実行をおこなわせる．ELSE 命令の右に続く命令行は $TEST 値が 0（偽）の場合にのみ実行され，$TEST 値が 1（真）の場合は ELSE の右にあるどんな命令行も実行されない．ELSE は引数をとらない
FOR	F I=7, "BD"	FOR はその行にあるあとのすべての命令行を繰り返し実行することを指定する．FOR には 3 種の書式がある．第 1 は式だけの型であり，左端のローカル変数に 7 または "BD" などの

命令語	例	説　　明
FOR	F　J=2:4:20 F　K=1:1 F　L=10:−1:0,13,15:1	値を割り付けられる．第2は範囲を指定する書式で，ローカル変数に初期値が割り付けられ，指定された増分ずつ次々に値を増しながらついに別に指定された値に達するかまたはそれをこえるとあとの命令行の実行が終る．第2の例では初期値は2であり，増分は4で最終値は 20 である．この際右に続く命令行が実行されるのはJの値が 18 であるときが最後であることに注意．第3の書式は初期値と増分値のみが与えられる“無限”ループ型で，第3の例に示される．この“無限”ループは QUIT または GOTO 命令が FOR 命令の右に続く命令行の中で実行されたときはじめて中断される．上記の3書式は混合してもよい．たとえば最後の例では範囲指定書式（負の増分を用いている）が示されている．次いで初期値式が2個あり，最後は“無限”ループとなる．FOR ループは“無限”ループのみでなく，すべての型で QUIT または GOTO 命令が実行されると中断されることに注意．また，QUIT は FOR 命令が入れ子になっているときはもっとも手近な FOR ループのみを終らせ，他方 GOTO はすべての活動している FOR ループを終らせることに注意
GOTO	G　NEXT G　^ROUTINE G LINE^ROUTINE G　@VARIABLE	GOTO はプログラムの実行の制御を永久的に他所に移動させる．これはもとにもどってくることはない．その移動先は同一ルーチンの1つの行（たとえば，G　NEXT）か，別のルーチンのはじまり（たとえば，G　^ROUTINE），または別のルーチン内の特定の行（たとえば，G　LINE ^ROUTINE では実行の制御は ROUTINE というルーチンの LINE という行へ移動する）である（制御をもとにもどそうとするならば，DO 命令を使うべきであることに注意）
HALT	H	まず引数をもたない LOCK（後述）と，この作業が占有していたすべての装置の CLOSE が実行される（それらが明確に宣言されている必要はない）．その後，現在の処理課程が中止される．HALT は引数をもたない
HANG	H　3 H　TIME	HANG は引数が指定する秒数（たとえば3とか TIME のような）だけ実行を中断する．もし引数が0より小さい値のときは実行は0秒中断する．引数をもたない HANG は HALT と見なされることに注意
IF	I I　A=3 I　B="B",A>4 I　C=4!(C=5) I　$D(X),X=4	IF は条件付きの実行を提供する命令である．それが引数なしの形で用いられると，実行は $TEST の値によって左右される．もし $TEST の値が 0（偽）ならば IF の右側の行の残りの部分は実行されない．逆に $TEST の値が 1（真）ならばふつう右側にくる命令文の実行が起こる．もし1つの引数があるときは（たとえば，A=3）その引数が真であるか偽であるかを評価し，その結果 $TEST の値がそれぞれ1か0に設定さ

命令語	例	説明
		れ，その後 IF 命令は引数なしの形と同じようにして実行される．1個の IF に続く引数の並記は，それらの1つ1つの引数を伴う IF 命令の並記のように扱われる．結果的にはこれらの引数が “and” でつながれた場合と同じである．ただし異なる点は並記の場合1つでも偽となる引数に出あうとその後の引数の解釈もその行の実行もただちに中止されることである
KILL	K K　A, NEW, ^P(3) K　(SAVE, A)	引数をもたない KILL は添字付き変数を含むすべてのローカル変数を消去する．1個の引数が指定されると，引数で名前を指定されたローカル変数やグローバル変数がすべての下位節とともに消される．第2の例では，ローカル変数 A と NEW，およびグローバル配列 ^P のうち第1レベルの添字が3である節がすべて消去される．“排他的 KILL” は KILL 命令の特別な形で，KILL の引数として添字をもたないローカル変数（たとえば，SAVE や A）を括弧に入れて指定する．この場合これらのローカル変数とそれらの下位節以外のすべてのローカル変数が消去される
LOCK	L L　^MUG L　(A, B, ^G(4))	LOCK は第一義的には名前のついた情報源（本来はグローバル変数）を更新する際の混乱を避けるためソフトウエア上の慣習として用いられる．引数を実行するたびに LOCK はそれまでにもっていた排外的な情報源占有権をすべて解除する．もしも名前をもつ情報源を指定すれば，それの一時的な占有権が確保される．もし他のユーザーがその指定した情報源の占有権をもっている場合，このユーザー（つまり LOCK 命令を実行しようとしているユーザー）は，その情報源を占有できるようになるまで待たされる．無駄に待つことを避けるために “時間制限” があるが，“時間制限” は LOCK の引数のどれにでも，前述したように付けることができる（複数の情報源の名前は括弧に入れて，L(A, B, ^G(4)) のように書く必要があり，1個の情報源の名前の場合は括弧が必要でないことに注意）．LOCK は既存の変数の定義や “略式グローバル指標” に対しては何の影響も与えない
OPEN	O 4 O 7 : (devparms)	OPEN 命令は装置の排他的な占有権を得るのに使用する．これはルーチンが対話している “現用装置” に影響は与えない．また装置名のあとに言語装備に特有な装置パラメータを置くことができる．占有権は CLOSE 命令を実行することによって解除される．“時間制限” はその装置を OPEN できないときにその命令を無効にするためにどの引数に付けてもよい
QUIT	Q	QUIT は，1つの FOR，DO または XECUTE 命令に対する出口点を定義する．この命令語は引数をもたない

命令語	例	説明
READ	R VALUE, X R *A R "NAME", N	READ は現用装置からのデータの入力を指定し，指定された変数にその入力を文字列値として割り付ける．引数の中で変数名の前に "*" 印があると1文字をあらわす数字コードが入れられる．文章型や書式制御文字（後述）も READ の引数として許され，それらは現用装置に出力される．入力を終えるまでの時間は READ の引数に付けた "時間制限" で指定できる
SET	S NEW="BIG" S X="B", ^GO(1) =9 S (I, J, K)=1	SET は変数に値を明確に割り付けるための一般的な方法である．複数の変数名を括弧に入れ SET 命令と割り付け符号（"="）の間に入れると，括弧の中に指定されたすべての変数に割り付け符号より右にある値が割り付けられる
USE	U 4 U MT+2: (devparms)	USE は指定された装置（たとえば装置番号4とか装置 MT+2）を入出力装置の現用装置または装置状態として指定する．装置指定子は装備特異的である．USE 命令の引数として指定しうる前に OPEN 命令の実行によってその装置の占有権が確保されていなければならない．CLOSE や OPEN 命令のように装備特有の装置パラメータを装置名のあとに置くことができる
VIEW	言語装置によって異なる	VIEW は標準 MUMPS 内で言語装備特有な情報をしらべたり変更したりするためのアクセス点を用意する
WRITE	W "HELLO" W A, B, Z W *13 W !?5, NAME W "AVG=", SUM /TOT	WRITEは現用装置にデータや書式制御の出力を指定する．変数値の前に "*" 印のついたものが引数となる場合は，その整数によってあらわされる番号のコードをもった1つの文字が現用装置に送られる．この文字の装置に対する作用は装置に依存し，また装備特有なものである
XECUTE	X "S A=3"	XECUTE はプログラムの実行中に生起する MUMPS コードを解釈する手段を備える．XECUTE 命令の引数はそれぞれ MUMPS コードの1行（ラベルや行開始指標のない）として解釈される
Z	言語装備によって特有な仕様をもつ	文字Zではじまる命令語の名前は標準 MUMPS に対する装備特有な言語拡張のために確保されたものである．MDC の検討を通じて普遍的な仕様としてとり上げられたものが将来の標準 MUMPS の中に追加される

関　数

どの関数もすべて "$" 文字を頭において構成されており，また "$" に続く第1文字のみに省略してもよいユニークな名前をもっている．その機能をあ

らわす文字があって次いで括弧（　）の中に1つもしくはそれ以上の式を入れることになる．第1番目の式は一般に検査されたり操作されたりする文字列ないし数字を入れている．そのあとにくる式はどれも関数の作用の質を規定する．関数の“$”記号の次にくる第1文字に含まれていない文字は将来の標準 MUMPS の利用のために確保されている．

ある種の関数では用いる式を限定している．このような限定を示すために，次のようなコードを用いてみよう．どの場合でも“n”は式の用いられる位置をあらわす．

En	限定をうけない一般の式
In	整数に評価される式
Tn	真偽値式
Nn	数値に評価される式
L	ラベル
VN	変数名

表 19·3 関数一覧表

関数名と構文規則	説　明	例	結　果
$ASCII (E1) $ASCII (E1, I2)	$ASCII は文字列内の1文字を選び，その文字の ASCII コードを返す．E1 で対象となる文字列を指定する．もし I2 が明確に指定されていなければ，その値は1としてみなされる．文字列 E1 の第 E2 番目の位置にある文字が取り出され，ASCII コード表上のその文字をあらわす整数に翻訳される	$A("ABC") $A("ABC", 1) $A("ABC", 2) $A(X, 2) $A(" ")	65 65 66 66 ただし X="ABC" とする −1
$CHAR(I1) $CHAR(I1, I2) $CHAR(I1, I2, ...)	$CHAR は，整数の集合をそれらの整数が ASCII コードである文字列に翻訳する．I1, I2 などは整数として解釈される式である．負の整数の場合は空文字を返す	$C(65) $C(66) $C(65, 66)	"A" "B" "AB"
$DATA(VN)	$DATA は変数が定義されているかどうかを示す整数を返す．VN は問題のローカル変数，またはグロ	$D(Y) $D(Y) $D(^A(1, 5))	Yが未定義ならば 0 Yが定義されており下位節をもたなければ 1

関数名と構文規則	説　　明	例	結　　果
$DATA(VN)	ーバル変数の名前である．もしVNが未定義ならその値として0が返され，VNは値をもつがそれに関係する下位配列がなければ1，VNは値をもたないが少なくとも1つの下位節があれば10，VNが値と少なくとも1つの下位節の両方をもてば11が返される 注）$D(VN)>9になる場合がある条件下で起こるが，この際にVNに何も下位節がないことがある．たとえば，すべての下位節がKILLされたときである		0，1，10，または 11
$EXTRACT(E1, I2) $EXTRACT(E1, I2, I3)	$EXTRACT は文字列 E1 内の位置 I2 の文字から位置 I3(I3が指定されなければ I2) または文字列の終端のうち位置番号の若い方までの文字または部分文字列を返す	$E("ABC", 2) $E(VAR, 3, 5) $E(VAR, 3, 999) $E(S, 90) $E("XYZ",-6, 2)	"B" "CDE" VAR="XXCDE". "10A" VAR="B910A". " "(Sの文字数が90以下のとき "XY"
$FIND(E1, E2) $FIND(E1, E2, I3)	$FIND は文字列 E1 上での部分文字列 E2の位置番号 +1を示す整数を返す．第3の式 I3がある場合，これはE1 内でE2 をさがすのをI3番目の文字からはじめることを指定する．もしI3 がなければ捜査は最初の文字からはじめられる．I3（または1）からはじめるときのE1の文字が空文字列かまたは E2を含まないときは結果は 0（ゼロ）である．もし E2が空文字で（ただしI3または1からはじめる E1の文字ではない）あれば結果は I3である	$F("ABC","B") $F("ABCABC", "A", 3)	3 5
$JUSTIFY (E1, I2) $JUSTIFY(N1, I2, I3)	2個の引数の $JUSTIFY は式 E1 の値を式 I2の長さの文字領野に右詰めにした形の値を返す．しかしI2 の値が E1の長さに等しいかまたは短いときは E1の値がそのまま返される．3個の引数があるときは最初の引数（N1）の数値解釈がおこなわれ，この数値は小数点以下 I3 桁で四捨五入がおこなわれ（必要に応じてゼロをつける），その値は	$J(39, 3) $J(39, 4, 1) $J(-5/3, 0, 0)	"39" "39.0" "-2"

関数名と構文規則	説 明	例	結 果
	E2 の長さの文字領野に右詰めにされる		
$LENGTH (E1)	$LENGTH は文字列 E1 に含まれる文字の数，すなわち文字列の長さを返す	$L("ABC") $L(X) $L(X)	3 4 ただし X="ABCD" のとき 0 ただし X=" " のとき
$NEXT (VN)	$NEXT は VN に指定された変数より大きく，かつ最小の添字の値を返す．もし VN より大きい添字がなければ結果として −1 が返される	$N(A(1)) $N(A(−1)) $N(^MUG (4))	2 ただし A(2) が定義済み(ポインターまたはデータがある) −1 6，ただし^MUG(6) は定義されており ^MUG (5) が定義されていない
$PIECE(E1, E2, I3) $PIECE(E1, E2, I3, I4)	$PIECE は指定された文字列（E1）をしらべて，その中の特定の境界文字列（E2）によって両側を挟まれた部分文字列を取り出す．その取り出される部分文字列は第（I 3-1）番目の境界文字列のすぐ次の文字から第 I 4 番目（I 4 が指定されなければ第 I 3 番目）の境界文字列のすぐ前の文字までである	$P("A. BX. Y", ".", 2) $P("76 A666", "6", 1, 4) $P(ST, ".", 2) $P(ST, ".", 1)	"BX" "76 A66" " "，ただし ST にピリオドが含まれていないとき ST ただし ST にピリオドが含まれていないとき
$RANDOM (I1)	$RANDOM は 0 から（I1-1）の範囲の 1 つの整数を返す．その範囲のどの整数の返される確率も等しい	$R(2) $R(100)	0 または 1 0 から 99 までの範囲の 1 つの整数
$SELECT(T1 : E1, T2 : E2, . . ., Tn : En)	$SELECT は 1 対になった式の連続のうちから左側の式を 1 個ずつ左から右に向って評価してゆき真偽値が 1（真）であるものが見つかるまでそれを続け，真偽値の 1 である左の式と対の右側の式を返す．どの式の対でも左の真偽値式（Tn）と右の "結果として返す" 式（En）との間はコロンで分ける 式の対は，1 個またはそれ以上でよいが，真偽値式のうちの少なくとも 1 つは真（1）でなければならない	$S(A=3: 5, 1: 0) $S(X=7: "HI", 1: "BYE") $S(Z=1 : 1, Z<0: 0, 1: 2)	A の値が 3 ならば 5 を返す X が 7 という値でないときに "BYE" を返す Z が負の値のとき 0 を返す
$TEXT(L1) $TEXT (L1+I2) $TEXT(+I1)	$TEXT はそのルーチン内の指定された行のコードをそのまま，行開始記号や，ラベル名を含んだまま	$T(LINE) $T(A+3)	LINE というラベルのついた行の内容を返す A とラベルされた行の次

関数名と構文規則	説明	例	結果
	返す．行はラベル名（L1）によっても，ラベルからのオフセット数（L1+I2）によっても，またルーチンのはじめからのオフセット数（+I1）によっても示される．指定された行が存在しない場合は空文字列（" "）が返される	$T(+2)	から数えて3行目の内容を返す ルーチンの最初から2行目の内容を返す
$VIEW（引数構文は装備者が指定）	$VIEWは言語装備者の選択による装備に特有関数であって言語装備に特有なデータを得るために用いる	—	—
$Z_	標準 MUMPS に含まれない関数はすべて"$Z"ではじめなくてはならない	—	—

特殊変数

特殊変数はそれぞれが"$"文字を頭におき，それと次の第1文字1個でその特殊変数名の省略をすることができるユニークな変数名である．特殊変数の値はどんな一般的な式の中に用いてもよいが，特殊変数に対してユーザーが値を割り付けることはできないことに注意．ここで指定されていない"$"文字の次の文字は将来の使用のために確保されている．

特殊変数	説明
$HOROLOG	$HOROLOG は日付と時間をあらわす2つの部分からなる1つの文字列である．その2つの部分はコンマによって分けられている．はじめの部分は1840年12月30日を0として数えた日数であり，第2の部分は当日午前0時からの秒数である．"0,0"が1840年12月31日の翌朝の第1秒目までの値である
$IO	$IO は現在使用中の入出力装置の識別記号をもっている
$JOB	MUMPS のおのおのの処理過程にはそれぞれ独自の作業番号が付いている．それは正の整数でそれが

	$JOB の値である．
$STORAGE	$STORAGE はユーザー パーティションでまだ使える文字数をあらわす
$TEST	$ TEST は最後に評価された真偽値が入っている．この値は，最後に実行された引数をもつ IF，あるいは時間制限をもつ OPEN，LOCK，READ によってその値が割り付けられる
$X	$X は現用装置が印刷端末ならキャリッジが，ビデオ端末ならばカーソルが水平位置で第何番目の欄にあるかを示す．最初の欄は第0欄，次は第1欄などと定義されている．すなわち各行のはじめは $X が 0 である．そしてその行に文字が1つ置かれると $X は1となる
$Y	$Y は現用装置上の垂直位置をあらわす．第1行は行 0，第2行は行1，などと定義されている．改頁によって $Y は0と置かれ，改行するごとに1ずつ増える
$Z__	標準 MUMPS にない特殊変数は "$Z" ではじまらねばならない

p103

書式制御

次の文字はデータ表示の際の書式制御として用いられる．

?　これは？のあとにくる式の整数値によって指定された欄へタブ（右方移動）させることを指定する．たとえば，?XはX欄にタブを移動させる．前記 $X のように MUMPS は最初の欄を0欄，第2の欄を1欄，などとして扱うことに注意

?X のXが $X と同じときは，タブの移動は起こらない

!　これは復帰改行またはそれと同様な他の行更新操作を指定する

♯　　これは改頁または頁更新を指定する

行とルーチン

MUMPS のルーチンは順序立った一連の行からなる．行は自由な書式によらず，行開始標識（装備者によって指定される）と，ときにはそれに先行する行ラベルをもち，そのあと0個またはそれ以上の命令語および関連する引数，そして任意にコメントをつけたものとして定義できる．行が複数の命令行からなるときは，これらを正確に1個のスペースで分けるが，引数をもたない命令語のあとにくる命令行はそれらの間に正確に2個のスペース（ここで1個のスペースは空の引数を意味する）を置く．ラインの中に置くコメントは最後の引数ないし，引数なしの命令語のあとにセミコロン（；）を置いてそのあとに書く．セミコロン以下はコメントとして扱われ実行されない．行ラベルは大文字アルファベットか“%”記号ではじまり，第2文字以後は 26 個の大文字アルファベットまたは 10 個の数字のうちのどれかから成る（移送基準を満足するプログラムでは行ラベルの英文字はすべて大文字でなければならない）．行ラベルはまた，すべて数字でもよい（すなわち整数そのものでよい）．ラベルの文字数に制限はないが，はじめの8文字が識別に用いられる．行ラベルはルーチン内では同じものが2個あってはならない．

〔移送基準に関する理由から，MUMPS 文字列の長さに関する現在の制限は255文字である．この255文字という長さの制限はプログラムとして記憶される行ラベル，行開始標識，命令，コメントをすべて含んだ1個の行についても適用される．1個の MUMPS ルーチンの中での行の数には明確な制限はないが，ルーチンの大きさはローカル変数および一時的な結果をすべて含めて4000文字をこえてはならない〕

1個の行に複数の命令語を一群として含むことは単に便宜上のことではないことは重要で，FOR, IF, ELSE などの命令語はその行全体を意義をもった1単位としてとり扱うことに注意．

付録　装備特有の機能

（この表は MUMPS 機種提供者ないし，特定機種のユーザーが，そのシステムの装備特有の機能について記載し，または配布すべき項目のおもなものを示す）

行開始標識(ls)：

行終了標識（eol）：

READ 入力終了標識：

命令語（Zではじまるもので，直接モード，間接モードの用法別を含み記載）：

関数（$Zではじまる）：

特殊変数（$Zではじまる）：

BREAK 命令の引数：

VIEW 命令：

装置指定式：

装置パラメータ：

邦 訳 付 録

MUMPSの文字列の分類と訳語

- 標準 MUMPS の全文字列概念
  - データ（ASCII 文字の1つ又はいくつかを連結したもの，又は何もなし（空文字列 empty string））
    - ↑↓（$TEXT）（間接指定）（XECUTE）
  - プログラムコード
    - 式以外のもの……命令語（command word），ラベル（label），注釈文（comment），etc.
    - 式（expression）
      - 値をもたず式子と式子の関係を与えるもの：二項演算子……+，−，*，／，＼，#，>，<，]，[，=，?，&，!，etc.（binary operator）
      - 値をもつもの：式子（expression atom）（一項演算子（unary operator +，−，'）を含む）
        - 値が変化するもの
          - 記憶域参照（storage reference）
          - 関数（function）
          - 特殊変数（special variable）
        - 値が変化しないもの 定字列(literal)
          - 定文字列（string literal） 引用符ではさまれた文字列（空文字列を表わす場合は引用符を2つ続ける）
          - 数字列（numeric literal）
            - 整数字列（integer literal）
            - 整数字列以外のもの……小数，指数表示

用　語　索　引

【ア行】

【カ行】

【サ行】

【タ行】

【ナ行】

【ハ行】

【マ行】

訳者紹介

若井一朗（わかい　いちろう）

1956年　名古屋大学医学部卒業
1957年　Fulbright 留学麻酔学専攻
1962年　Fellow of American College of Anesthesiology
1963年　名古屋大学大学院医学研究科外科学専攻
1964年　医学博士
1968年　鳥取大学助教授
1969年　同大付属病院中央手術部長
1973年　社会保険中京病院麻酔科部長兼コンピータセンター長
1975年　日本 MUG 発起人，米国 MUG 運営委員，日本 MUG 会長
1976年　マンプスシステム研究所所長同付属クリニック院長
1977年　名古屋大学医学部講師
1978年　日本マンプスユーザーズグループ書記長
1980年　MEDINFO 研究会評議員

MUG 編／コンピュータ言語
MUMPS 入門　　NDC 549.92

昭和 55 年 10 月 30 日　初版発行

（定価はカバーに表示してあります）

訳者　若井一朗
発行者　梅川雪夫
発行所　日刊工業新聞社
東京都千代田区九段北一丁目 8 番10号
（郵便番号 102）
電話東京（263）2311（大代表）
振替口座東京 9-186076

印刷所　壮光舎印刷株式会社
製本所　三和製本所

定価 2500円

3034－507200－5719